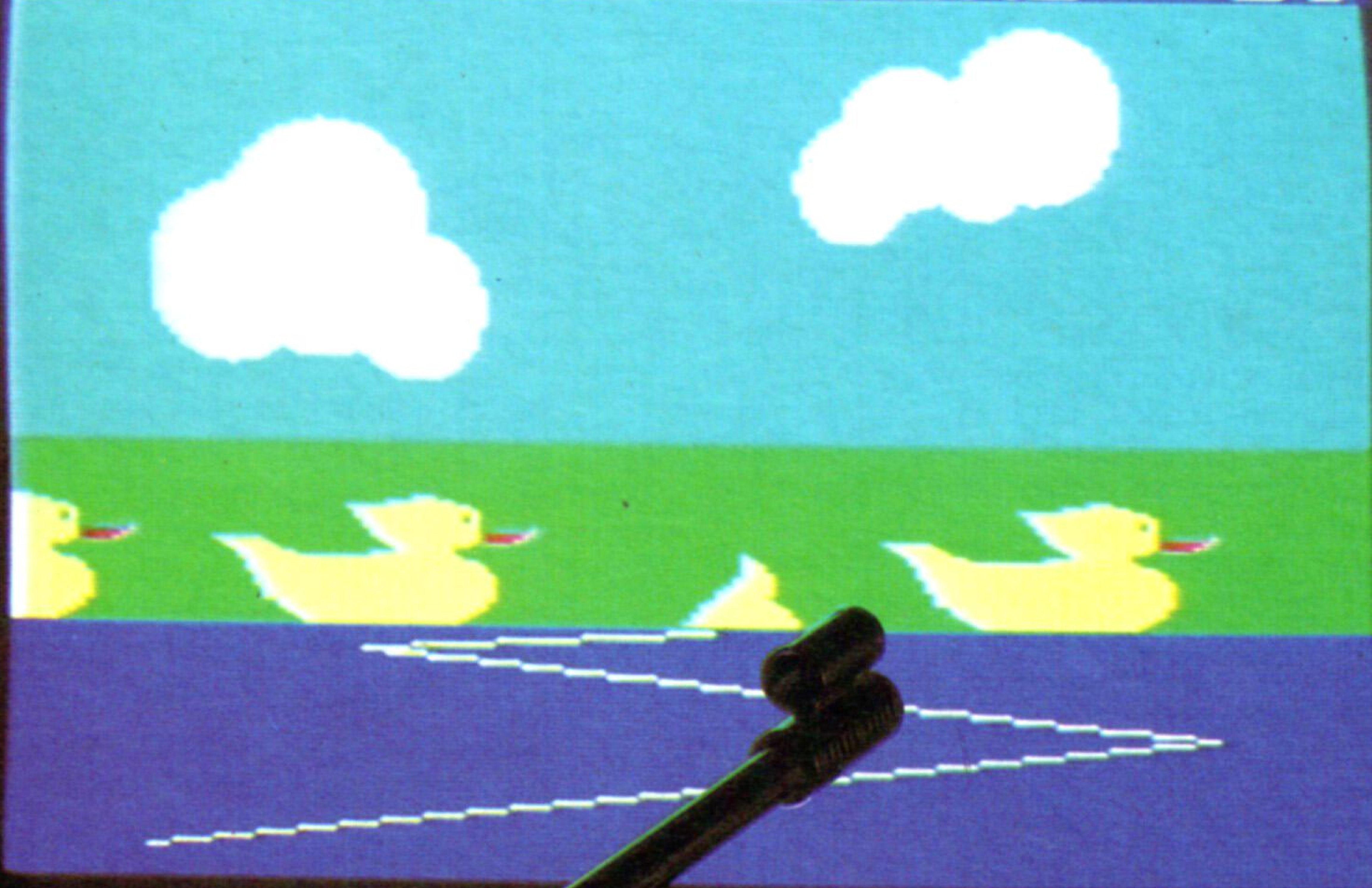
150 ptas.

MI COMPUTER

36

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR





Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona, y comercializado en exclusiva por Distribuidora Olimpia, S.A., Barcelona

Volumen III - Fascículo 36

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Jesús Nebra

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti. A. Cuevas, F. Blasco

Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, Barcelona-8
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-85822-94-3 (tomo 3)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 198409
Impreso en España - Printed in Spain - Septiembre 1984

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, Madrid-34.
Distribuye para Argentina: Viscontea Distribuidora, S.C.A., La Rioja 1134/56, Buenos Aires.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio Blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Ferrenquín a Cruz de Candelaria, 178, Caracas, y todas sus sucursales en el interior del país.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 16 690 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 087 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 3371872 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Distribuidora Olimpia (Paseo de Gracia, 88, 5.°, Barcelona-8), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Distribuidora Olimpia, en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

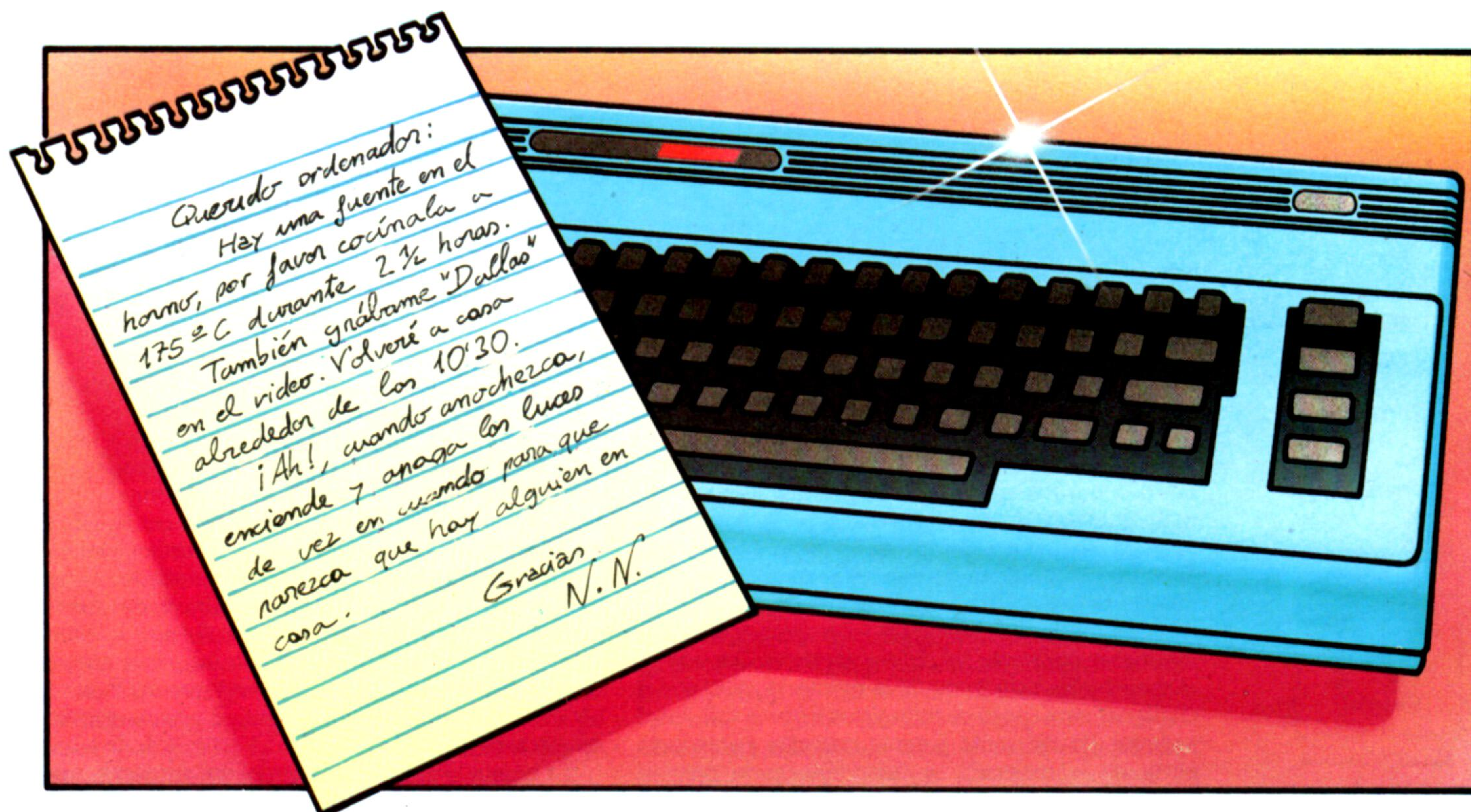
No se efectúan envíos contra reembolso.

Editorial Delta, S.A.



Su seguro servidor

Ya es posible hallar en el mercado componentes que permitan crear dispositivos que realicen las más diversas tareas



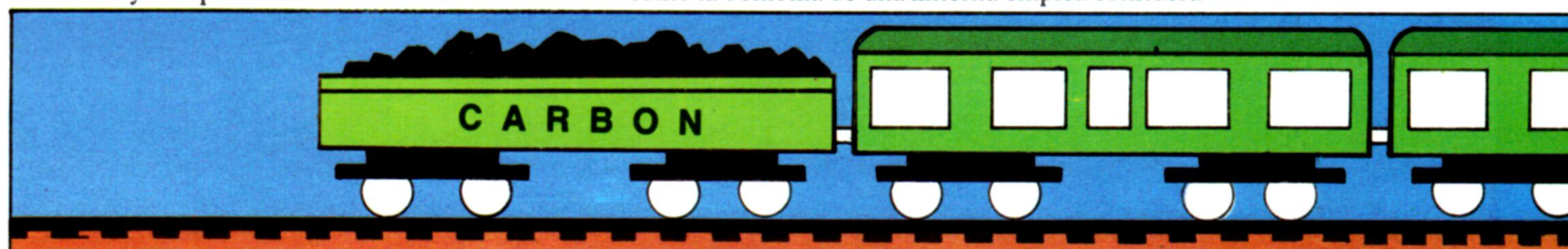
Mike Brownlow

La justificación para montar un micro con el fin de que abriera las cortinas por la mañana o de que regara las plantas cuando uno estuviera de vacaciones podría ser, simplemente, porque hacerlo resulta divertido. Y no hay nada de criticable en realizar algo porque a uno le agrada hacerlo.

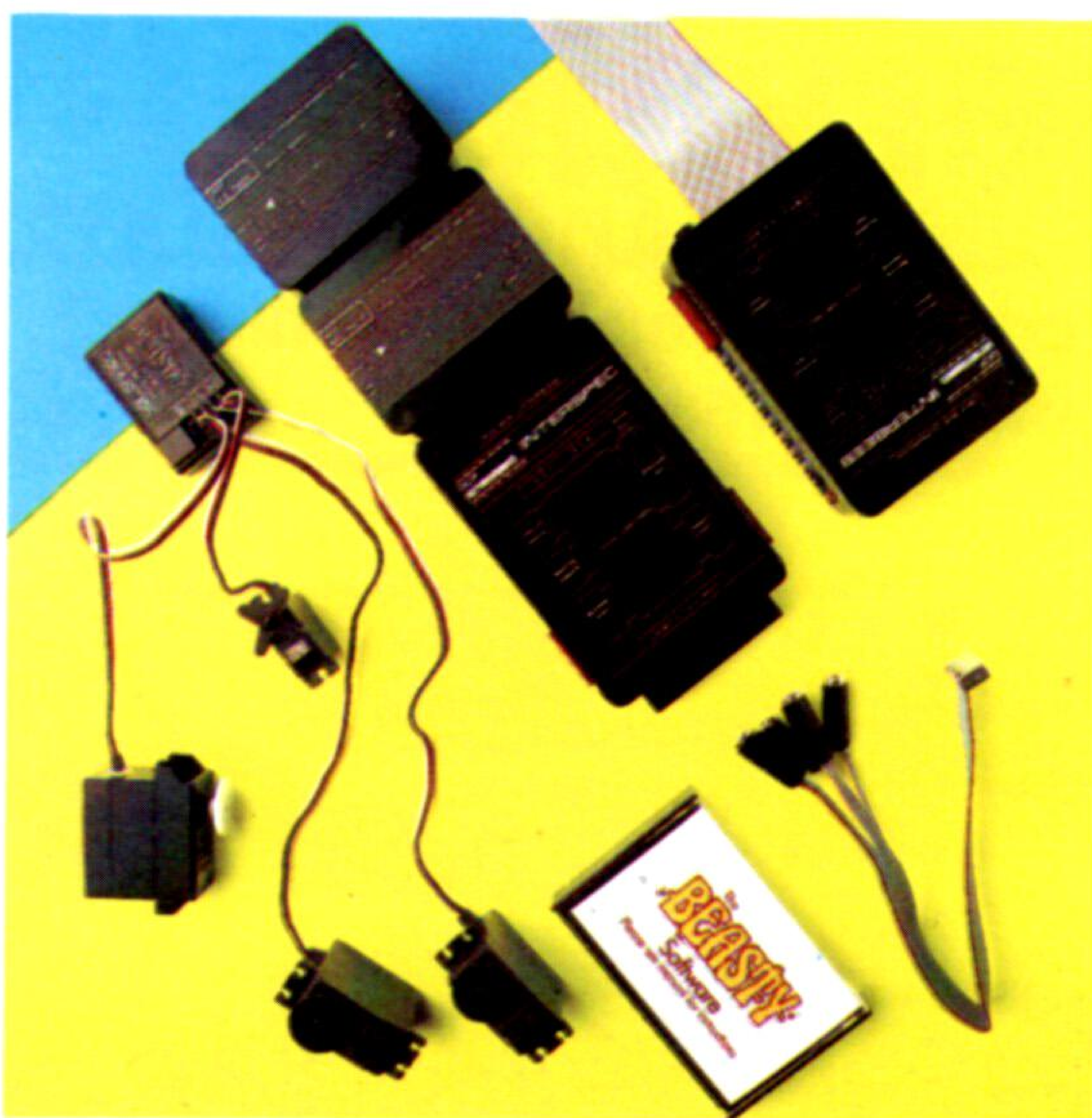
En la actualidad, construirse sus propios dispositivos periféricos se podría considerar como "juguetear" con artefactos de fabricación casera, pero a la larga esta actividad podría resultar productiva. Mucha gente piensa que los robots controlados por ordenador y otros dispositivos serán pronto una parte integral en nuestras vidas (del mismo modo que los ordenadores se han convertido en algo cotidiano en apenas cinco años), y, por consiguiente, las habilidades aprendidas ahora podrían tener un valor incalculable en el futuro. Después de todo, empresas informáticas gigantescas, como Apple y Atari, nacieron en los garajes particulares de personas que trabajaban informalmente y a su aire con artefactos y componentes electrónicos.

Cualquier sistema controlado por ordenador precisa de varios elementos. Obviamente, son primordiales la propia máquina y el artículo que esté bajo su control. También es indispensable que haya algún medio en virtud del cual el ordenador le transmita al dispositivo los mensajes de control, y el software que permita que el ordenador decida cuáles han de ser esos mensajes. Y esto es sólo la mitad del proceso. El ordenador suele precisar de algún sistema para medir el efecto que está teniendo su control, con el fin de realizar los ajustes necesarios. Esto se conoce como *realimentación* (*feedback*) y sin ella el ordenador sería tan inútil como un conductor de automóvil con los ojos vendados.

Todos los sistemas controlados por ordenador dependen de las señales eléctricas para realizar su control. Lamentablemente, las señales eléctricas que se utilizan en el interior de un ordenador son demasiado imperceptibles como para poderlas utilizar directamente. Hasta un dispositivo tan pequeño como la bombilla de una linterna emplea considera-



Control remoto
Conectar un ordenador a otros aparatos proporciona toda clase de posibilidades de operación automática bajo el control de un programa. El ordenador puede responder a horas preprogramadas o reaccionar ante hechos tales como un descenso de la temperatura o la desconexión de una alarma antirrobo



Conexión de control
Las interfaces aptas para usos de control se venden ya hechas para ser usadas con muchos micros personales, sobre todo con el BBC y el ZX Spectrum. Estas unidades normalmente son unidades de conmutación basadas en relés. El ordenador puede conectar o desconectar un aparato determinado, y posteriormente enterarse de si el piloto del mismo se ha encendido o no

Ian McKinnell

blemente más energía que cualquiera de las partes que se hallan dentro de un ordenador; de modo que se necesita algún medio de traducir los mínimos voltajes del interior de un ordenador a voltajes más potentes (y algunas veces éstos pueden serlo tanto como los voltajes de la red eléctrica). Normalmente esto se cumple en una serie de etapas. La primera se realiza dentro del propio ordenador. Éste necesita enviar de alguna manera señales al mundo exterior. Esto se hace eligiendo una porción de la memoria del ordenador y reservándola exclusivamente para este fin. El microprocesador enviará mensajes a esta parte de la memoria igual que a cualquier otra, pero cuando éstos lleguen allí serán tratados de diferente forma. Por esta razón esta parte de la memoria se denomina *toma para el usuario* y la información almacenada en ella se puede leer electrónicamente desde fuera del micro.

Algunos micros poseen una toma para el usuario como estándar, otros disponen de ella como una opción. En sí misma, la toma para el usuario se puede utilizar para encender y apagar LED (diodos emisores de luz), pero la mayoría de los sistemas prácticos necesitan además otros componentes. Tal vez lo más útil es añadir unos pocos componentes electrónicos, más una pequeña fuente adicional de energía eléctrica, para permitir controlar los relés. Los *relés* son esencialmente interruptores que pueden encender y apagar corrientes eléctricas relativamente grandes y que, no obstante, se pueden controlar mediante pequeñas corrientes eléctricas. Proporcionan una de las mejores maneras de convertir los impulsos eléctricos que se emplean en un ordenador en corrientes útiles. Una de las primeras tareas de cualquiera que experimente con el control por ordenador es crear un sistema que permita usar relés, que pueden controlar variados mecanismos, desde motores eléctricos a trenes en miniatura y coches controlados por radio.

La mayoría de los relés que emplean los aficionados sólo pueden hacer frente a la clase de dispositivos que funcionan con pilas. Esto representa un amplísimo espectro para la mayoría de proyectos. Pocas personas se encontrarán con la necesidad de conmutar aparatos que se conecten a la red. Dado que la electricidad de la red es sumamente peligrosa, sólo se deben utilizar productos comerciales que hayan sido probados con todo rigor, y no hay muchos disponibles. Quien trabaja con voltajes pequeños puede elegir entre construir y comprar unidades de conmutación de relé ya hechas que se conectan directamente en un ordenador.

La conmutación de la red permite que la máquina controle calefactores, luces intensas y docenas de otros artículos domésticos. También permite que el ordenador actúe como un reloj para encender el televisor a la hora del programa favorito, o para encender y apagar las luces de la casa transcurridos determinados lapsos para disuadir a los ladrones de penetrar en la casa.

En la actualidad, el ordenador debe estar conectado directamente a las unidades que esté controlando, y esto es más bien una limitación. Varias empresas están desarrollando productos para superar este inconveniente. Estos productos funcionan permitiendo que los cables que distribuyen por toda la casa la electricidad de la red transporten también datos. El sistema tendrá un ordenador en una habitación enviándole señales a unidades *esclavas* dispuestas por toda la casa y conectadas a enchufes comunes. El ordenador puede enviar mensajes individuales a cada unidad indicándole que se conecte o se desconecte. Cualquier aparato doméstico se puede enchufar en la unidad y ser controlado a través del ordenador.

Ya hemos dicho que casi todos los sistemas controlados por ordenador necesitan algún tipo de realimentación para medir cómo está funcionando el sistema. Podría parecer que cuando un ordenador se está limitando a encender y apagar las luces no hay necesidad de realimentación. Todavía sería mucho mejor si el ordenador pudiera saber cuándo está oscuro afuera y encender las luces en respuesta a esa información. También sería útil que el ordenador pudiera detectar cuándo alguien entra en una habitación y encenderle la luz. Si un ordenador está controlando un calefactor, necesita realimentación para determinar las alteraciones de temperatura de la habitación y poder así mantener ésta dentro de unos límites prefijados.

Existen dos tipos de señales de realimentación. Algunas sólo pueden estar encendidas o apagadas, sin estados intermedios. Una señal de este tipo podría ser un interruptor que reconozca si una ventana está abierta o cerrada, o si ha sonado el timbre de la puerta o no. Las tomas para el usuario son capaces de leer este tipo de *señales on/off*.

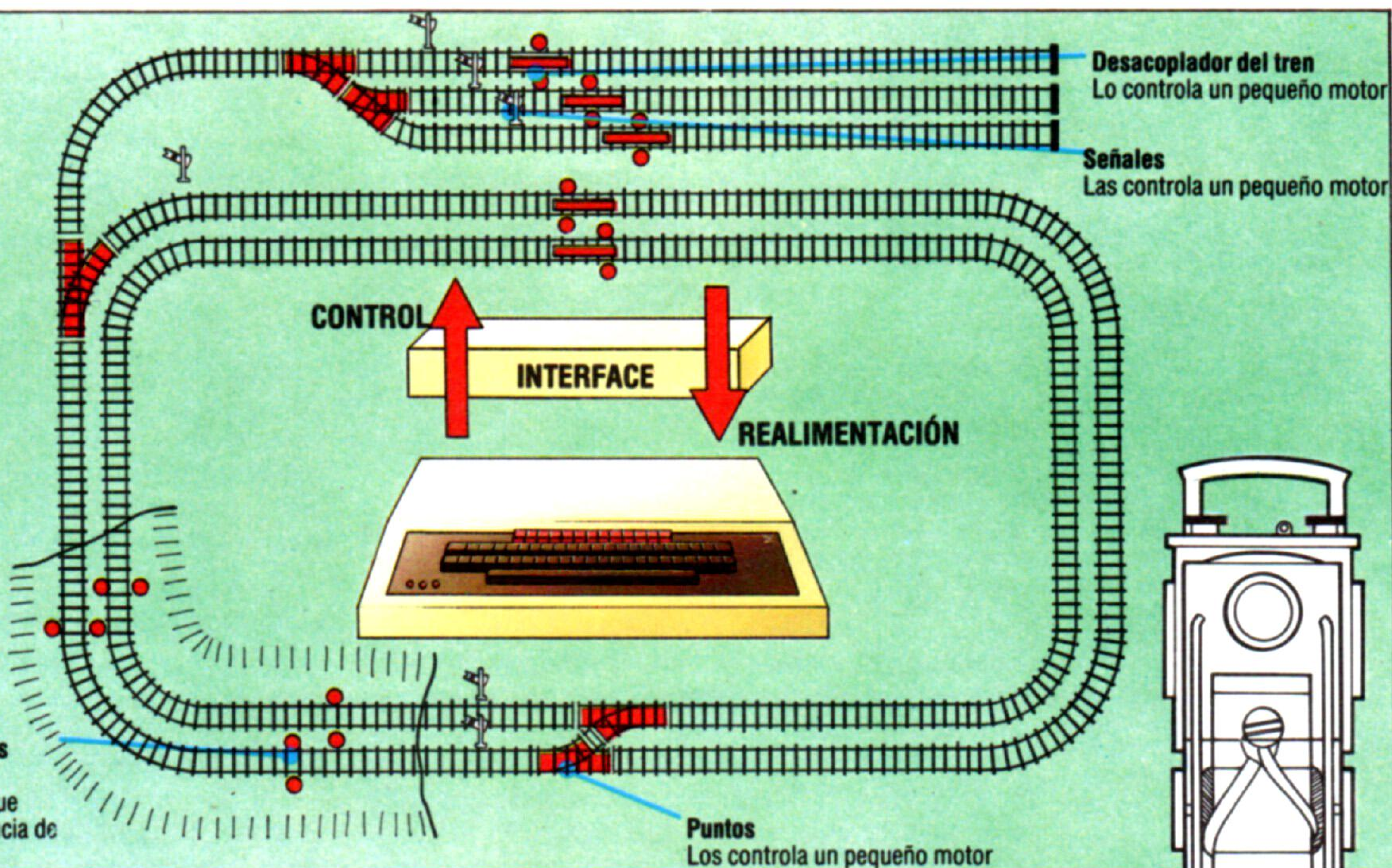
Un tipo de realimentación más útil, pero ligeramente más complicado, proporciona una *señal ana-*





El cerebro del ferrocarril

Controlar cualquier tipo de aparato con un micro (desde un tren en miniatura hasta una casa entera) implica la misma técnica básica. Se establece un bucle en el cual el micro envía señales de control al dispositivo en cuestión a través de una interface. Éstas controlan servomotores, luces, etc. El dispositivo devuelve entonces información de realimentación mediante interruptores accionados por el peso del tren o células fotosensibles. Este bucle de realimentación permite que el ordenador maneje con precisión el aparato



Kevin Jones

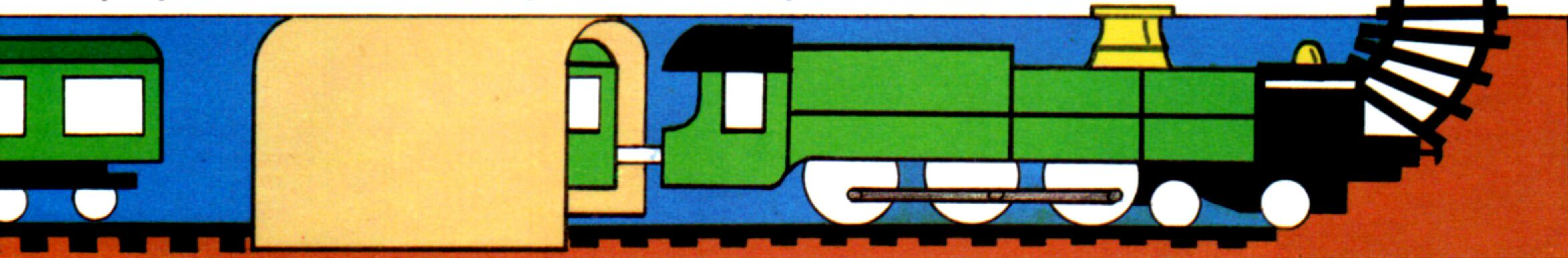
lógica. Una señal de esta clase puede ser de una escala de valores y, por tanto, se puede utilizar para medir el calor que hace, qué distancia se ha desplazado o ha girado un objeto, cuánto pesa algo o qué voltaje está dando una pila. El dispositivo que acepta este tipo de señal se denomina *convertidor de analógico a digital* (A/D): toma la escala variable de valores del equipo que se está controlando (la señal analógica) y la convierte a una forma digital que el ordenador puede comprender.

La disponibilidad de realimentación significa una gran diferencia en cuanto a lo que se puede hacer bajo control del ordenador. Si un motor está accionando una rueda, el ordenador podría calcular cuánto ha girado ésta en un cierto tiempo. Esto podría no funcionar, sin embargo, si se aplicara una carga sobre la rueda o si las pilas que activasen el motor se estuviesen agotando, porque entonces la rueda giraría más despacio. Un sensor óptico podría informar al ordenador cada vez que la rueda completara una revolución: de este modo, el ordenador podría controlar perfectamente su giro.

Algunos tipos de motores eléctricos hechos especialmente para usos de control llevan incorporada una cierta clase de realimentación. Ello significa que el ordenador envía una señal que les indica que se muevan hasta una posición determinada y el motor sigue funcionando hasta que llega a la misma. Existen dos tipos principales de esta clase de motores: *motores paso a paso* y *servomotores*. Un motor paso a paso puede girar continuamente, como un motor normal, o se puede detener en cualquier posición. No obstante, carece de potencia, de modo que sólo puede hacer frente a cargas pequeñas. Los servomotores son potentes pero sólo pueden girar un ángulo pequeño, por lo general de poco más de 90º. Esto a menudo se convierte en un movimiento de tipo empuje-tracción. Tanto los motores paso a paso como los servomotores necesitan unidades de control especiales para hacerlos funcionar con ordenadores, y éstas no están disponibles para muchas de las marcas de ordenadores personales menos usuales. Los servomotores se emplean en muchos brazos-robot. Es posible hallar en el mercado numerosos brazos-robot que se pueden conectar con ordenadores personales, pero son muy caros. Pueden construirse por un precio más reducido a partir de unos pocos servos.

La categoría final de dispositivos controlados por ordenador que vamos a considerar aquí requiere para su control un voltaje variable. Un ejemplo de ellos es un pequeño motor eléctrico que gire a distintas velocidades, según el voltaje que se le aplique. Un *convertidor de digital a analógico* (o D/A) es lo contrario de uno A/D: cambia las señales digitales que utiliza el ordenador a voltajes variables. Éste se podría emplear, por ejemplo, para producir sonido conectándolo a un altavoz.

Utilizar micros para controlar otros aparatos es como si uno escribiera su propio software. Se ha de combinar una idea con algún conocimiento técnico y mucho tiempo. A veces los resultados no están a la altura de los estándares comerciales, pero "hacerlo uno mismo" es, sin lugar a dudas, mucho más gratificante para el usuario que adquirir en el comercio productos de fabricación masiva.



Diseño gráfico

En este capítulo veremos cómo con instrucciones sencillas se pueden convertir gráficos matemáticos en originales diseños

Para trazar el gráfico de una expresión matemática se toma una escala de valores para una de las variables y se calculan los valores correspondientes para la otra variable. Con un gráfico simple como $Y = X^2$, podríamos elaborar esta tabla:

X	−5	−4	−3	−2	−1	0	1	2	3	4	5
Y	25	16	9	4	1	0	1	4	9	16	25

Dibujando los puntos en una hoja de papel cuadriculado y uniéndolos luego mediante una curva continua obtenemos la familiar curva en forma de collar. La curva es el gráfico de $Y = X^2$. Otra forma de ver la curva es pensar que es el camino que sigue un punto que se desplaza a lo largo de $Y = X^2$. Al camino se lo suele llamar *lugar geométrico*, y combinando distintos lugares geométricos podemos producir sorprendentes patrones. Para hacer esto analicemos un lugar geométrico: la circunferencia. La mejor forma de describir una circunferencia mediante una fórmula es ligeramente más complicada que el método anterior. Tanto X como Y se definen en función de una tercera variable o parámetro. Haciendo variar éste a través del juego de valores dado, se pueden calcular pares de X e Y. La circunferencia la da esta ecuación:



X = R SIN (I)
Y = R COS (I)

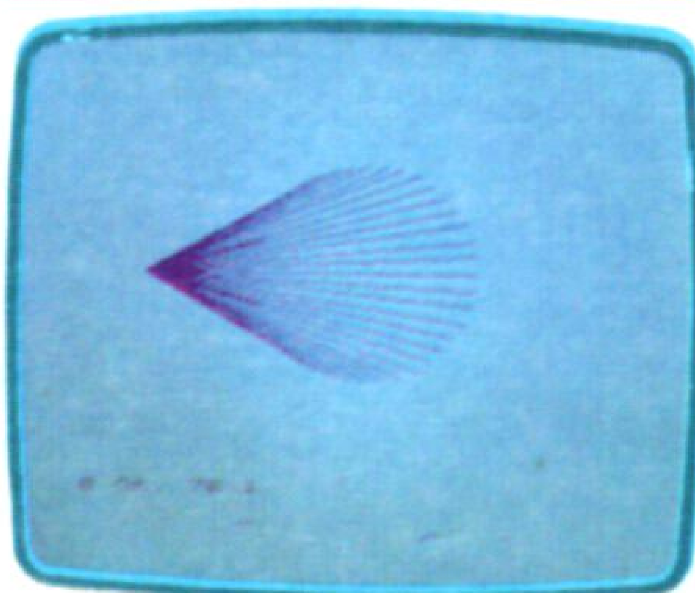
Si elaboramos un juego de valores para X e Y a medida que el ángulo I se gira en una circunferencia completa (de 0 a 360°), obtenemos el lugar geométrico de una circunferencia. El pequeño programa descrito a continuación para el Spectrum creará una circunferencia. (Véase "Complementos al BASIC" para las otras versiones de BASIC.)

```
10 REM Trazar circunferencia
20 LET xm=256: LET ym=176: LET xc=INT(xm/2): LET yc=INT
   (ym/2)
30 LET r=50
40 LET s=PI/20
50 FOR i=0 TO 2*PI STEP s
60 INK 2: PLOT xc+r*SIN (i),yc+r*COS (i)
70 NEXT i
```

Observe que cambiando el valor de STEP entre los puntos trazados se puede variar la definición de la circunferencia y alterar la velocidad a la cual se dibuja. La mayoría de las versiones de BASIC no son lo bastante rápidas como para dibujar circunferencias continuas a partir de muchos puntos individuales a una velocidad aceptable. Para superar este inconveniente, suele ser preferible utilizar un número de líneas rectas para unir los puntos de la circunferencia. Con un gran número de líneas rectas se puede conseguir un equilibrio razonable entre la velocidad y la uniformidad del dibujo. También puede emplearse esta fórmula para dibujar arcos y elipses. Para arcos, seleccione distintos valores de I. Para elipses, déle a R un valor diferente en la fórmula X al de la fórmula Y.

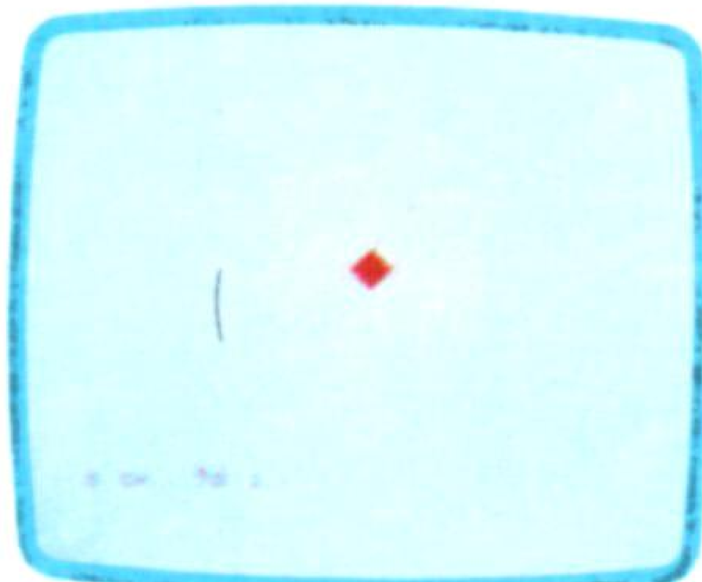
Nuestro primer patrón se crea dibujando una línea desde un punto fijo hasta todos los puntos que componen el lugar geométrico. Los siguientes programas posicionarán el punto en el centro de la circunferencia y a la izquierda de ella, respectivamente.

```
10 REM flor
20 LET xm=256: LET ym=176: LET xc=INT (xm/2): LET
   yc=INT (ym/2)
30 LET r=50
40 LET s=PI/20
50 FOR i=0 TO 2*PI STEP s
60 INK 2: PLOT xc+r*SIN (i), yc+r*COS (i): DRAW
   xc−(xc+r*SIN (i)), yc−(yc+r*COS (i))
70 NEXT i
```

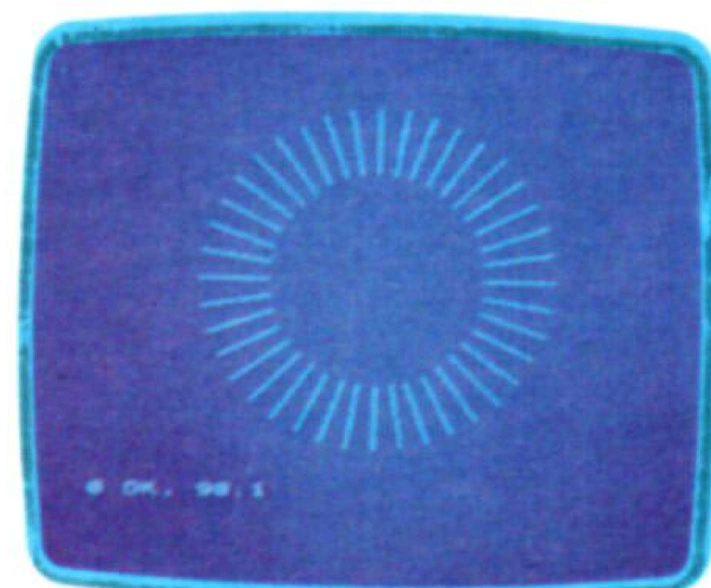


El siguiente paso consiste en utilizar un punto móvil en vez de uno fijo. Trazaremos dos lugares geométricos simultáneamente y dibujaremos líneas para conectar los puntos correspondientes en los dos caminos. El diseño más simple que esto creará es el de dos circunferencias anidadas unidas por numerosas líneas rectas.

```
10 REM Circunferencias anidadas
20 LET xm=256: LET ym=176: LET xc=INT (xm/2): LET
   yc=INT (ym/2)
30 LET r=50
40 LET s=PI/20
50 FOR i=0 TO 2*PI STEP s
60 LET x=xc+(r+30)*SIN (i): LET y=yc+(r+30)*COS (i)
70 LET p=xc+r*SIN (i): LET q=yc+r*COS (i)
80 INK 2: PLOT x,y: DRAW p−x,q−y
90 NEXT i
```

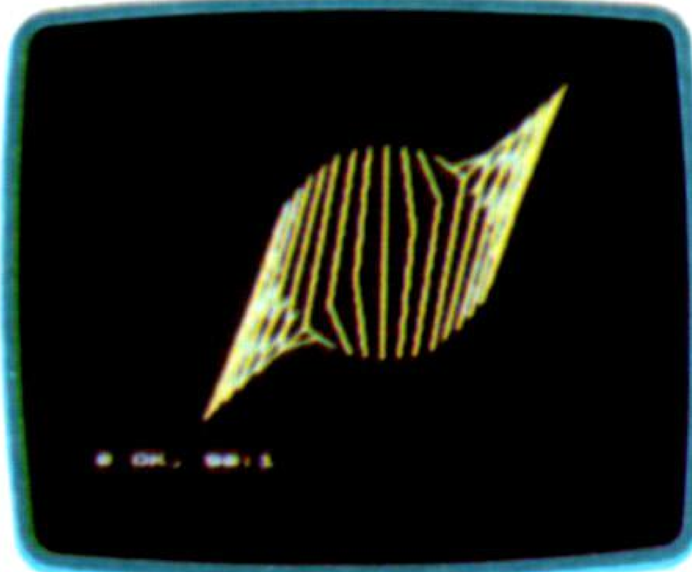


```
10 REM Circunferencia y punto fijo
20 LET xm=256: LET ym=176: LET xc=INT (xm/2): LET
   yc=INT (ym/2)
30 LET r=50
40 LET s=PI/20
50 FOR i=0 TO 2*PI STEP s
60 INK 2: PLOT xc+r*SIN (i),yc+r*COS (i): DRAW
   xc−100−(xc+r*SIN (i)),yc−(yc+r*COS (i))
70 NEXT i
```



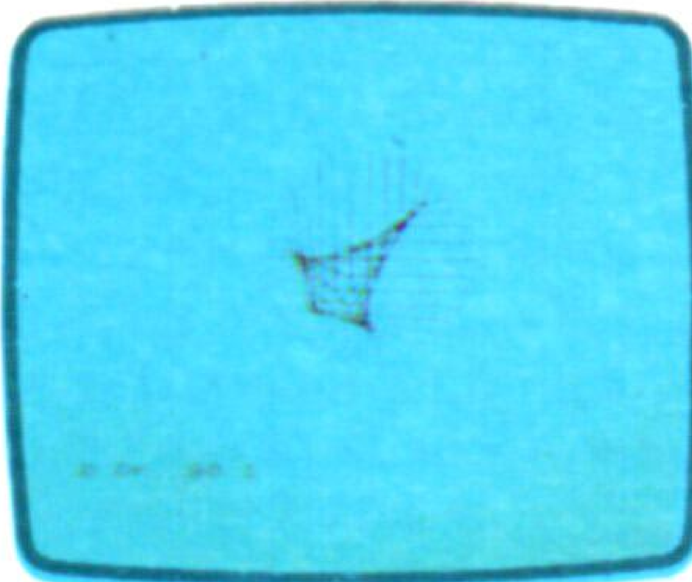


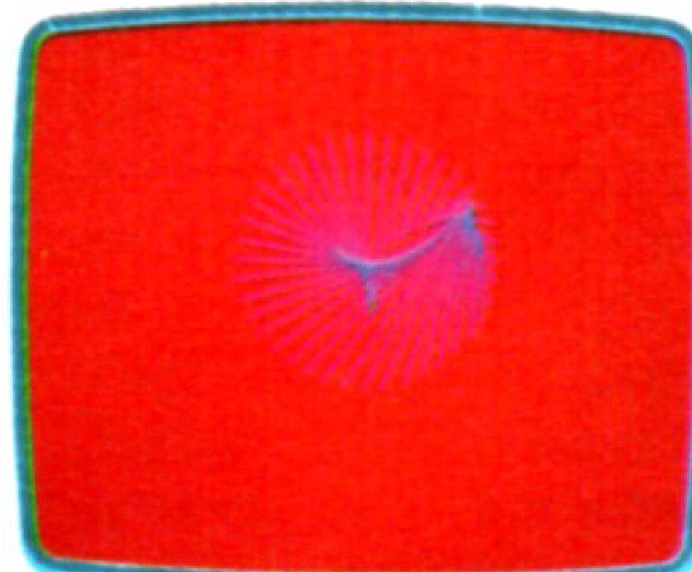
El programa desarrollado ahora incorpora información suficiente para dibujar cientos de patrones. Lo que hay que hacer es cambiar un lugar geométrico o ambos. Una variación sencilla consiste en permutar SIN y COS en una de las fórmulas. También podrían utilizarse potencias de SIN y COS (multiplicándolas juntas) para producir otros efectos.



```
10 REM Circunferencias retorcidas
20 LET xm=256: LET ym=176: LET xc=INT (xm/2): LET
   yc=INT (ym/2)
30 LET r=50
40 LET s=PI/20
50 FOR i=0 TO 2*PI STEP s
60 LET x=xc+ (r+30)*COS (i): LET y=yc+ (r+30)*SIN(i)
70 LET p=xc+r*SIN (i): LET q=yc+r*COS (i)
80 INK 2: PLOT x,y: DRAW p-x, q-y
90 NEXT i
```

Existen muchas variantes sobre estas ideas. Un libro de texto de matemáticas estándar le proporcionará las fórmulas para crear curvas alternativas. Sin embargo, probablemente encontrará que experimentar con su propia programación es un pasatiempo mucho más gratificante. Unas modificaciones al programa, pocas y sencillas, harán incluso que el ordenador haga la experimentación por usted. Aquí el programa final itera indefinidamente a través de un número de patrones generados al azar, aunque, naturalmente, no agota todas las posibilidades.





```
10 REM Circunferencia y SIN
20 LET xm=256: LET ym=176: LET xc=INT (xm/2): LET
   yc=INT (ym/2)
30 LET r=50
40 LET s=PI/20
50 FOR i=0 TO 2*PI STEP s
60 LET x=xc+SIN (i)*80: LET y=yc+(r+30)*SIN (i)
70 LET p=xc+r*SIN (i): LET q=yc+r*COS (i)
80 INK 2: PLOT x,y: DRAW p-x, q-y
90 NEXT i
```



```
10 REM Circunferencia y SIN*COS
20 LET xm=256: LET ym=176: LET xc=INT (xm/2): LET
   yc=INT (ym/2)
30 LET r=50
40 LET s=PI/20
50 FOR i=0 TO 2*PI STEP s
60 LET x=xc+r*SIN (i)*COS (i): LET y=yc+r*SIN (i)*COS (i)
70 LET p=xc+r*SIN (i): LET q=yc+r*COS (i)
80 INK 2: PLOT x,y: DRAW p-x,q-y
90 NEXT i
```

```
10 REM Patrones de circunferencias al azar
15 RANDOMIZE
20 LET xm=256: LET ym=176: LET xc=INT (xm/2): LET
   yc=INT (ym/2)
30 LET r=60: LET s=PI/20
35 CLS
40 LET c=INT (RND*4)+1: LET d=INT (RND*4)+1: LET
   e=INT (RND*4)+1: LET f=INT (RND*4)+1
50 FOR i=0 TO 2*PI STEP s
60 LET x=xc+r*SIN (i/c)*COS (i*d): LET y=yc+r*SIN (i/e)*
   COS (i*f)
70 LET p=xc+r*SIN (i): LET q=yc+r*COS (i)
80 PLOT x,y: DRAW p-x,q-y
90 NEXT i
100 IF INKEY$="" THEN GO TO 100
110 GO TO 35
```



Complementos al BASIC

Los programas listados aquí funcionarán en un Spectrum de 16 Kbytes y de 48 Kbytes. Sin embargo, convertir estas ideas y utilizarlas en otras máquinas es muy sencillo. Para ello, su micro necesita gráficos de alta resolución (preferentemente, al menos 256 × 176), un BASIC con coma flotante con las funciones SIN y COS, y una instrucción para trazar puntos individuales y dibujar líneas rectas.

Los primeros ajustes necesarios son XM e YM, los valores X e Y máximos que se pueden trazar en su máquina. Según las funciones que utilice, es posible que deba modificar los valores de otras variables del programa como R y S. Seguidamente debe asegurarse de que su micro esté en la modalidad de gráficos apropiada y seleccionar un color para el trazado. Por último, necesita una orden para dibujar líneas entre las coordenadas dadas en X e Y y en P y Q. En el Spectrum esto se ha de hacer con PLOT seguida de DRAW. La orden DRAW es complicada porque en el Spectrum siempre se relaciona con el primer punto dibujado, mientras que en este caso necesitamos dibujar una posición absoluta determinada. La mayoría de los micros posee una función para trazado de líneas absolutas y, por consiguiente, esta etapa es mucho más sencilla.

BBC MICRO Todas las modalidades del BBC utilizan un cuadriculado de la pantalla de 1280×1024 puntos para el trazado, en el que la instrucción MODE 0 produce resultados espectaculares. Utilice GCOL para seleccionar el color del trazado y MOVE y DRAW para dibujar las líneas.

DRAGON 32/64 La PMODE 4 del Dragon proporciona un cuadriculado de 256×192 apto para estos programas. Utilice SCREEN 1, 0 o SCREEN 1, 1 para seleccionar un fondo verde o bien marrón. La instrucción LINE se puede utilizar (LINE (X,Y)−(P,Q),PSET) para dibujar las líneas.

COMMODORE 64/VIC-20 Estas máquinas poseen gráficos de alta resolución adecuados, pero no las instrucciones apropiadas. Para ejecutar estos programas es necesario o bien suministrar las instrucciones para cada punto y línea o bien utilizar un cartucho de ampliación de BASIC, como el BASIC de Simon.

CAMPUTERS LYNX El Lynx es muy adecuado para esta clase de trabajo, puesto que posee una completa visualización para gráficos de 256×248 en ocho colores. Al igual que el Spectrum, no es necesaria ninguna instrucción de cambio de modalidad. Utilice INK para seleccionar el color del diseño y MOVE y DRAW para trazar las líneas.

ORIC 1/ATMOS HIRES activa la pantalla para gráficos de 240×200 del Oric. Las líneas se pueden dibujar utilizando CURSET para establecer el punto de comienzo (X,Y) y luego DRAW para trazar la línea. En el Oric, DRAW es una instrucción relativa, de modo que la orden DRAW ha de tener la forma DRAW p−x,q−y para funcionar.

Ideas de diseño

1) Volviendo al bucle simple para dibujar una circunferencia, hemos descrito cómo crear arcos y elipses mediante el mismo programa. Ahora vea si puede hallar una forma de dibujar espirales.

2) Intente utilizar otras funciones como SQR y TAN para generar lugares geométricos. Sepa que se deben emplear con cuidado porque tienden a generar números complicados. No obstante, debería ser capaz de producir resultados interesantes.

3) Realice versiones animadas de los programas. Utilizando matrices para registrar las últimas cinco líneas dibujadas, debería ser capaz de dibujar un grupo de cinco líneas persiguiéndose las unas a las otras alrededor de dos lugares geométricos.

4) ¿Por qué no intenta crear patrones basados en tres lugares geométricos? Utilice dos muy simples (p. ej., una circunferencia y una línea recta) para no crear dibujos demasiado confusos.

Continúa el serial

Ahora nos corresponde analizar cómo se crean, se accede a ellos y se actualizan los archivos secuenciales

Un archivo secuencial es un bloque de datos sólido contenido en un disco o una cinta y, como tal, existen limitaciones en cuanto a cómo puede ser accedido y actualizado. Para recuperar un ítem cualquiera, uno primero debe leer todos los datos precedentes. Para actualizar el archivo se suele hacer una copia del mismo hasta el punto que se necesita modificar, luego añadir los cambios al archivo nuevo y proseguir con la copia del archivo original inmediatamente después de las modificaciones.

Es importante comprender que la información ha de estar organizada de una forma adecuada dentro del archivo. La elección acerca del tipo de organización está en manos del programador y dependerá en cada caso de la aplicación. Si se trata de un archivo que contiene texto en castellano, es probable que sea suficiente una secuencia de códigos ASCII seguida de una marca de final de archivo. Sin embargo, si el archivo ha de contener una base de datos, como puede ser un catálogo de libros, entonces es necesario organizar la información de acuerdo con la estructura del catálogo. La forma normal de hacerlo consiste en dividir el archivo en *registros* y *campos*. Cada libro posee su propio registro (entrada) en el archivo y dentro de cada registro hay un número de campos (el título del libro, su autor, su editor, etc). En un archivo secuencial, estas divisiones se deben señalar utilizando caracteres especiales colocados entre los datos.

Esto se suele realizar utilizando un carácter de retorno del carro (código ASCII 13) para que actúe como delimitador entre los campos y registros. Dado que el archivo tendrá el mismo número de campos en cada registro, para el programa será fácil llevar la cuenta de dónde termina un registro y empieza otro.

Una vez que se ha creado un archivo secuencial, uno necesita ser capaz de acceder a él y actualizarlo. Las operaciones básicas sobre archivos son: recuperación de registros, adición de registros, eliminación de registros y modificación (edición) de registros. Los diagramas ilustran las diversas maneras de realizar estas operaciones con archivos secuenciales. Dado que usted sólo puede leer un archivo secuencial por orden y no puede cambiar libremente los datos que contiene, estas operaciones trabajan leyendo a través del archivo, creando una nueva copia a medida que se producen. Toda información que se va a cambiar se escribe en el nuevo archivo a medida que se crea. Por último, el nuevo archivo se convierte en el archivo en curso o actual y el antiguo es eliminado o bien se conserva como una copia de seguridad.

Estas sencillas técnicas constituyen la base de todas las rutinas de tratamiento de archivos secuenciales. No obstante, implican un supuesto fundamental acerca de las capacidades del sistema operativo: que éste pueda tener abiertos dos archivos al mismo tiempo con el objeto de leer en uno y escribir en otro simultáneamente. Esto no es posible en todos los sistemas de disco, y en los micros basados en cassette sólo es posible en aquellos que tengan acopladas dos grabadoras de cassette. Tanto la gama Grundy Newbrain como la Commodore PET disponen de interfaces para cassette gemelas por este motivo. Las máquinas con unidades de cassette individuales están limitadas a archivos suficientemente pequeños como para ser leídos por completo sobre la memoria y ser procesados allí.

Estos procedimientos de tratamiento de archivos también poseen un interesante efecto colateral. Después de que se haya introducido cualquier modificación en el archivo (ya sean adiciones, eliminaciones o modificaciones) se dispone de dos copias del archivo: una antigua, que es el archivo antes de que se lo actualizase, y una copia nueva con los cambios realizados. Es una práctica estándar en la gestión empresarial conservar ambos archivos, de modo que si algo le sucediera al nuevo siempre habría una copia que sólo estaría anticuada en un conjunto de cambios (o una generación). De hecho, se suelen conservar tres generaciones de cualquier archivo: al nuevo se lo denomina archivo *hijo* y al precedente se lo conserva como archivo *padre*. El archivo que se utilizó como base para el archivo padre se conoce como archivo *abuelo*.

Estas técnicas pueden trabajar con archivos que sean demasiado grandes como para caber en su totalidad en la memoria del ordenador, porque sólo una parte del archivo se está procesando en un momento dado. Sin embargo, con los archivos pequeños se puede obtener un rendimiento mucho mayor leyendo todo el archivo en matrices en la memoria y procesándolo allí. Todas las operaciones sobre archivos se pueden llevar a cabo a gran velocidad en la memoria antes de volver a escribir el nuevo archivo ya procesado en disco o en cinta.

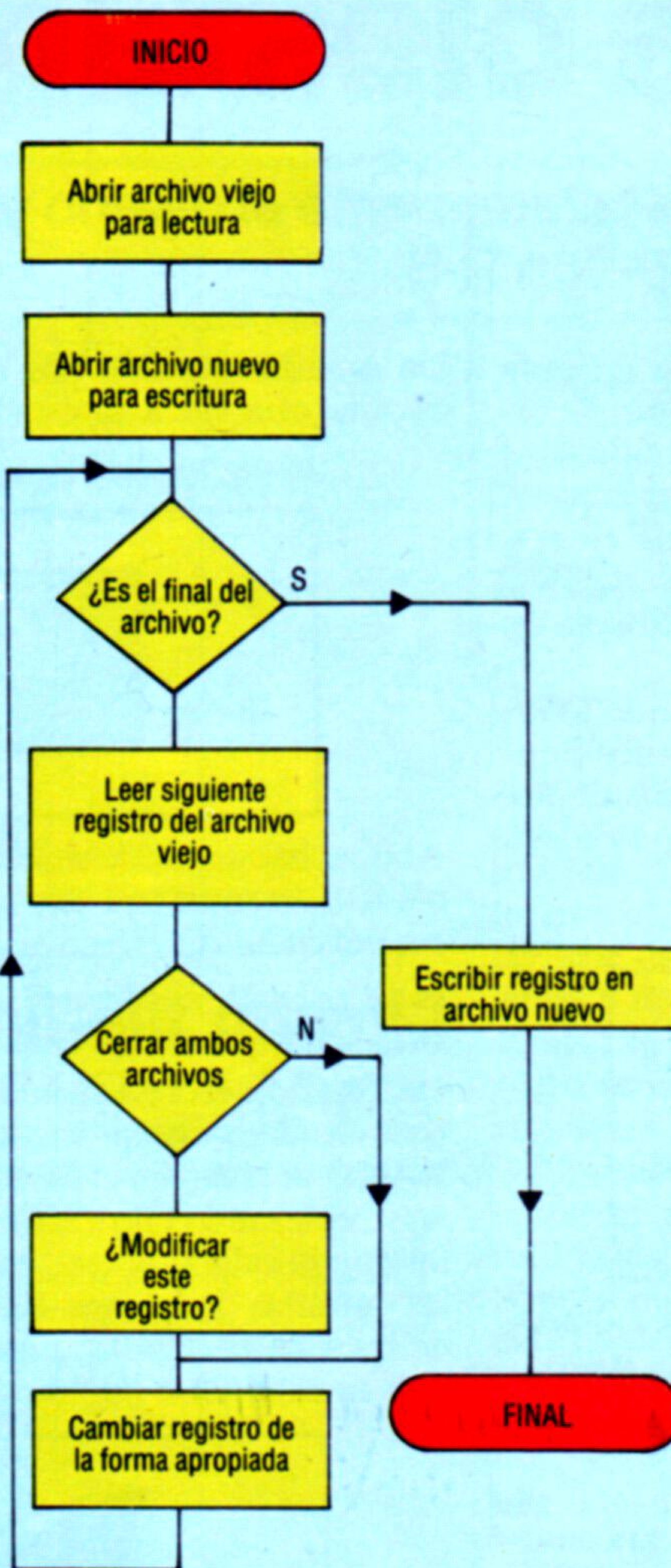
Este enfoque tiene un peligro fundamental: los cambios introducidos en el archivo sólo adquieren carácter permanente cuando la información se vuelve a escribir en cassette o en disco y, por consiguiente, los datos se podrían perder si el programa funcionase mal o si se apagara el ordenador durante la ejecución. Si usted utiliza programas que trabajan así, asegúrese de que se ha grabado una copia del archivo en curso antes de que el programa termine.

Con un poco de experiencia en tratamiento de archivos secuenciales comprobará que las técnicas pertinentes, a pesar de ser engorrosas, se basan fundamentalmente en el sentido común. En muchos sistemas pequeños, los archivos secuenciales son la única estructura de archivos proporcionada. Cuando analicemos los archivos de acceso directo, o aleatorio, descubriremos técnicas que complementan a los archivos secuenciales, proporcionando tanto un acceso como una actualización sencillos y rápidos.



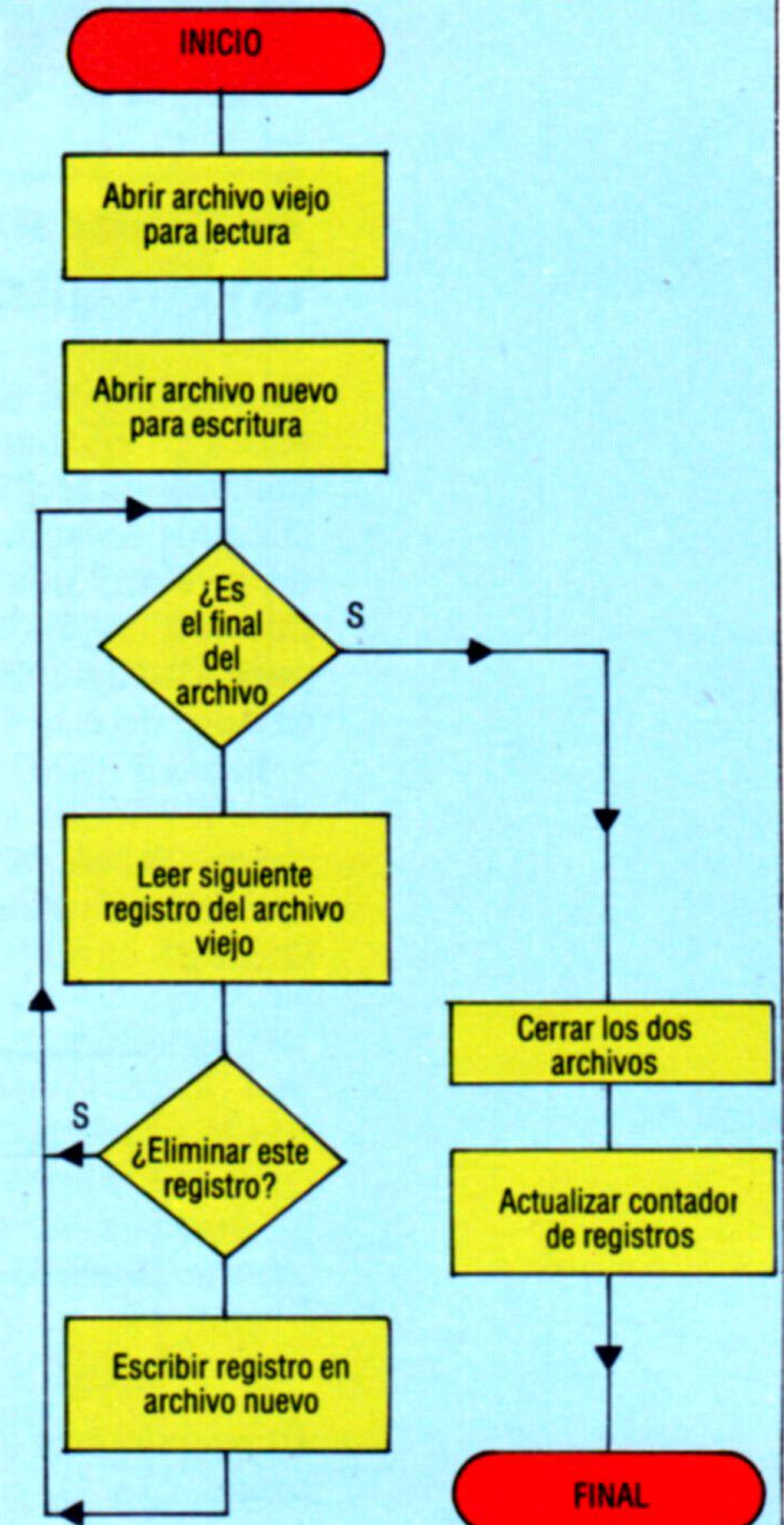
Modificación de un registro

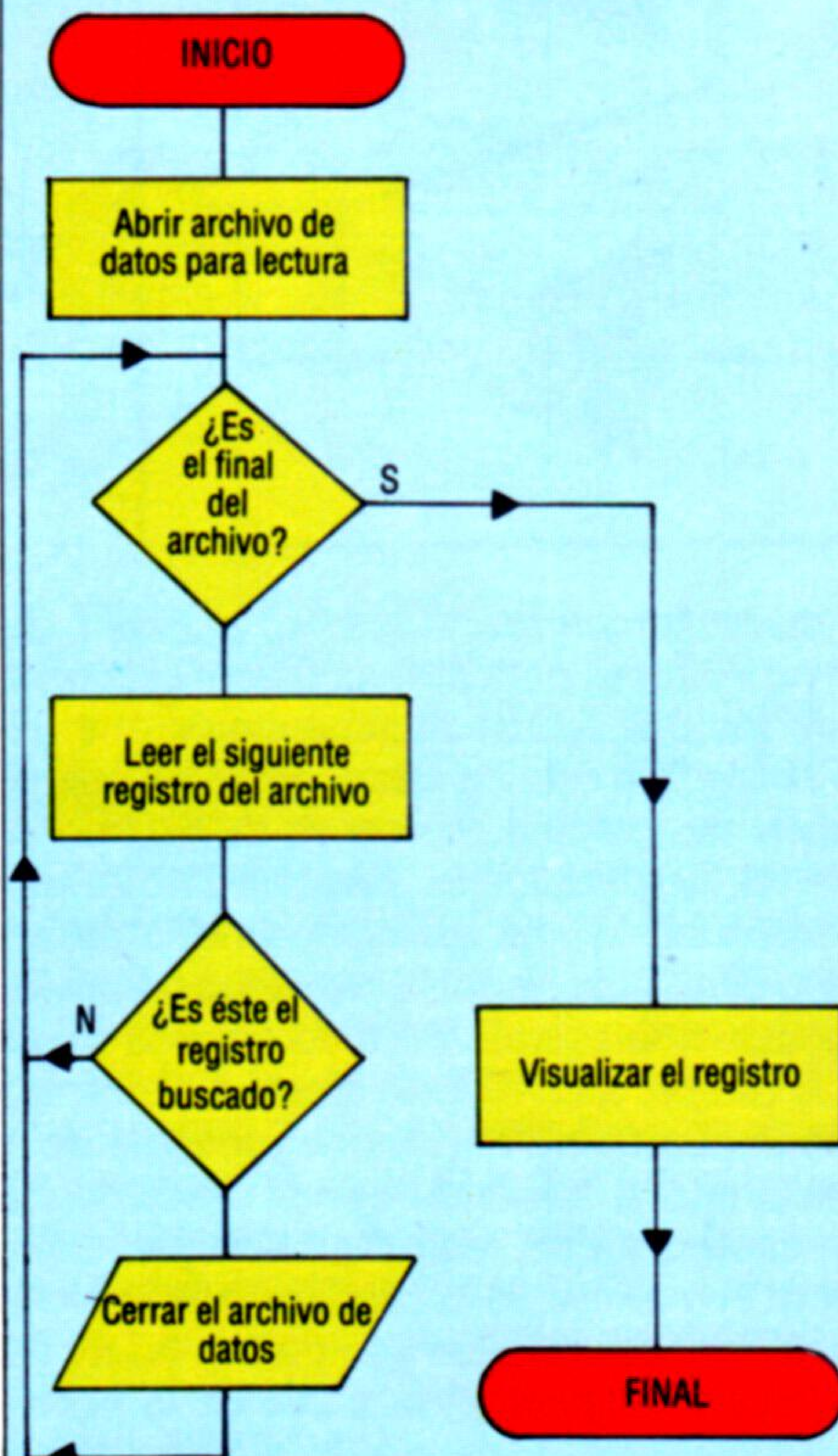
Modificar un registro (alterar la información que contiene) se consigue mediante una combinación de estos métodos. En primer lugar se deben copiar todos los registros precedentes en un archivo nuevo. Cuando se ha llegado al registro a modificar y se lo ha leído en RAM, éste puede ser modificado por el programa. El registro corregido se escribe en el archivo nuevo y todos los registros subsiguientes del archivo viejo se copian en el archivo nuevo a continuación del que se modifica. En una pasada a través del archivo se puede corregir cualquier número de registros



Eliminación de registros

Los registros se pueden eliminar de un archivo leyendo y copiando hasta el registro a eliminar. Este registro se lee entonces, pero no se escribe en el archivo nuevo. Por último, el resto del archivo viejo se lee y se copia en el archivo nuevo. En una sola pasada se puede eliminar cualquier número de registros. Al igual que cuando se añaden registros, es esencial que el contador de registros se actualice inmediatamente

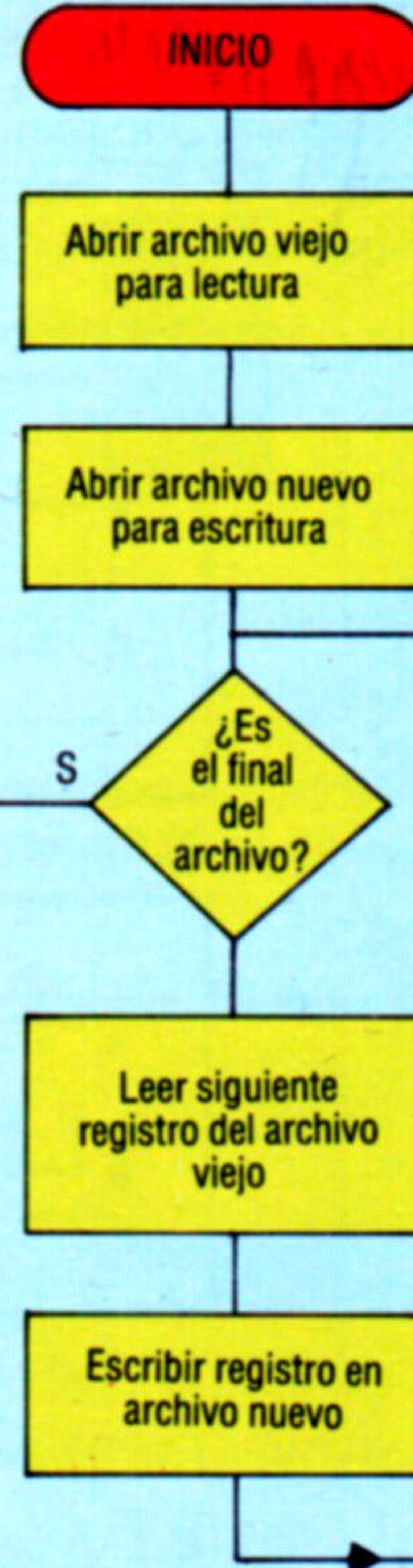




Recuperación de registros

Los archivos secuenciales no son muy idóneos para la clase de aplicación en la que uno coge registros antiguos del archivo. Cada vez que se busca uno de ellos, se ha de volver a leer el archivo desde el principio, lo que consume mucho tiempo. Si usted está buscando un registro determinado en una lista de nombres, entonces su programa simplemente pasará a través de un bucle, examinando cada registro y siguiendo adelante si no existe correspondencia entre el leído y el buscado. Si usted necesita un cierto número de registros, éstos se pueden leer uno después del otro pero sólo por el orden en que están grabados en el archivo. Por este motivo, a menudo los archivos secuenciales se clasifican por algún orden (p. ej., alfabéticamente) antes de almacenarlos





Adición de registros

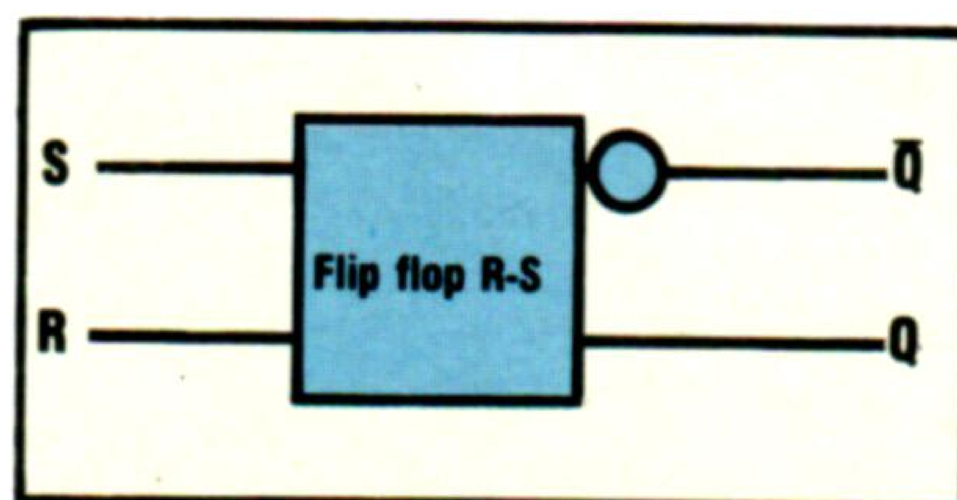
A un archivo se le pueden añadir registros de dos maneras. Algunas versiones de BASIC disponen de la instrucción APPEND (añadir) que permite que uno los agregue directamente al final de un archivo. Para añadir registros a un archivo sin disponer de esta instrucción, se debe leer todo el archivo y producir una copia del mismo en un archivo nuevo, y antes de cerrar el archivo nuevo, escribir los registros nuevos al final del mismo, cerrando luego ambos archivos. En los dos casos es necesario actualizar el contador de registros. Si éste se almacena con el archivo, la rutina de modificación deberá asegurarse de que el nuevo valor del contador se escriba en el archivo de inmediato, de modo que no exista la posibilidad de que esta información se pierda

Circuitos biestables

Esta vez examinaremos detalladamente el diseño y la función de un circuito biestable: el flip-flop R-S

Todos los circuitos que hemos analizado hasta ahora en el curso de lógica producen salidas determinadas al recibir ciertas señales de entrada. Los circuitos secuenciales, por otra parte, son capaces de producir una señal de salida uniforme en respuesta a un único impulso de entrada. Estudiemos, pues, con detenimiento el diseño y la función de un circuito de este tipo: el flip-flop R-S.

Existen diversos tipos de flip-flop, aunque todos ellos funcionan siempre en base a los mismos principios. Tal como se puede apreciar en la figura que aparece a continuación, el flip-flop R-S posee dos líneas de entrada y dos de salida.

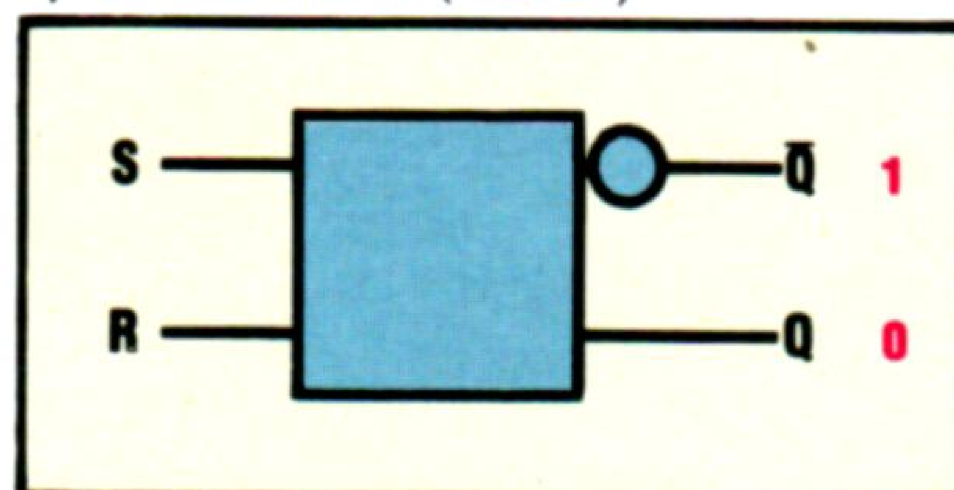


El circuito está diseñado de modo que las líneas de salida, Q y $\overline{Q}$, siempre sean opuestas. Es decir:

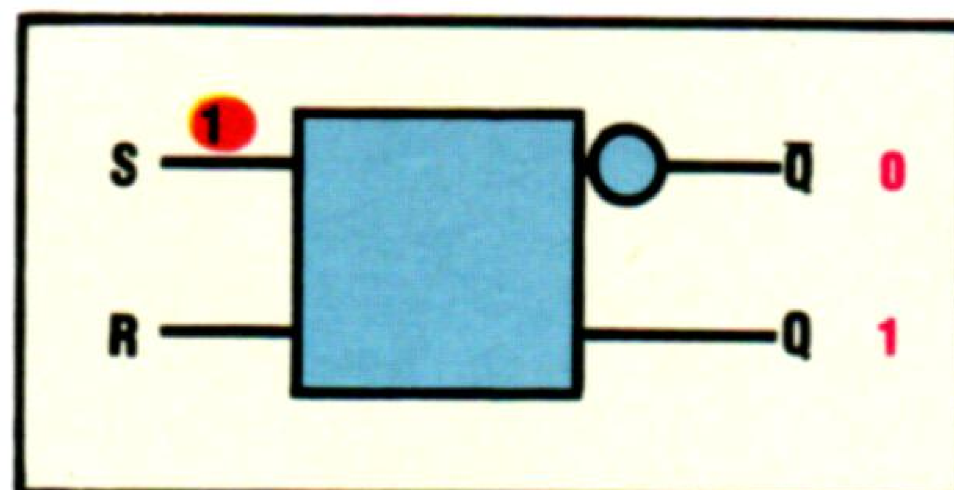
si Q=1 entonces $\overline{Q}$=0 (el estado SET)
si Q=0 entonces $\overline{Q}$=1 (el estado RESET)

Suponiendo que inicialmente el flip-flop está en estado RESET, entonces un impulso en la línea S hace que el circuito "salte" al estado SET.

1) Estado inicial (RESET)

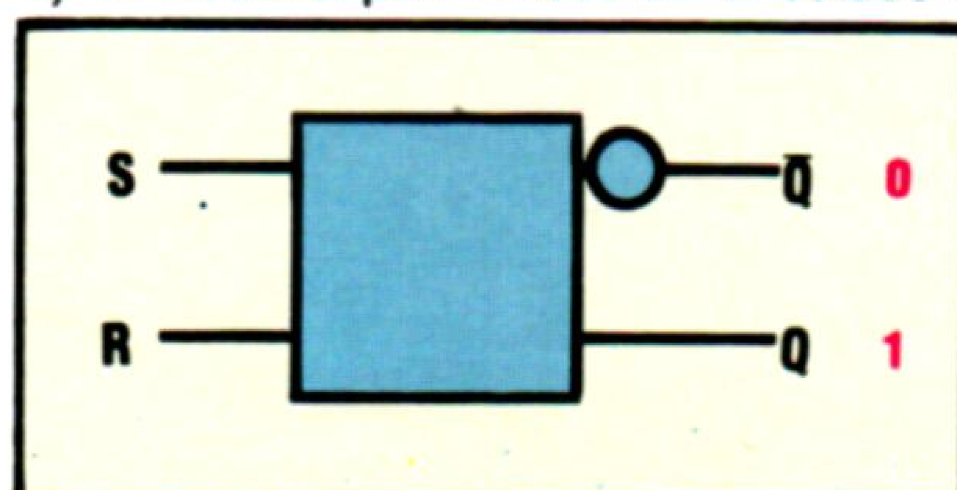


2) Un impulso en la línea SET



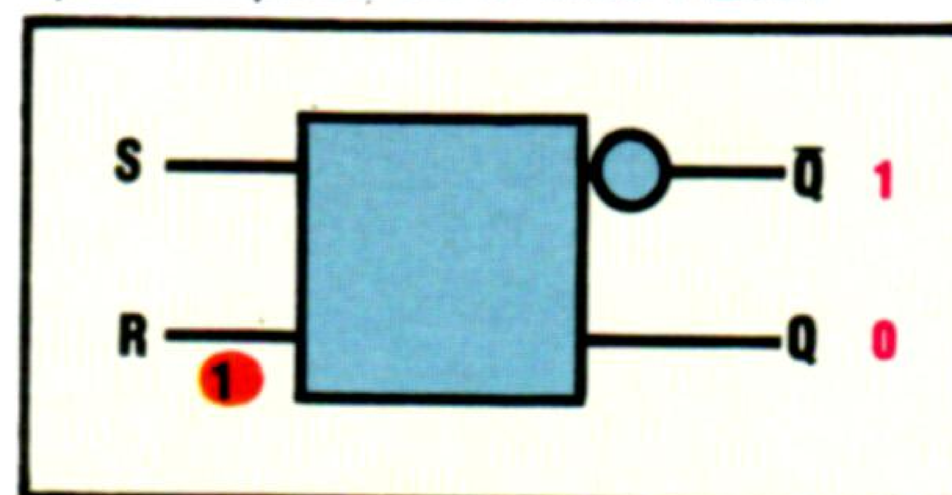
Cuando cesa el impulso de entrada en la línea S, el circuito permanece en el estado SET.

3) El circuito permanece en el estado SET



Un impulso enviado por la línea R hace saltar al circuito otra vez a su estado RESET original.

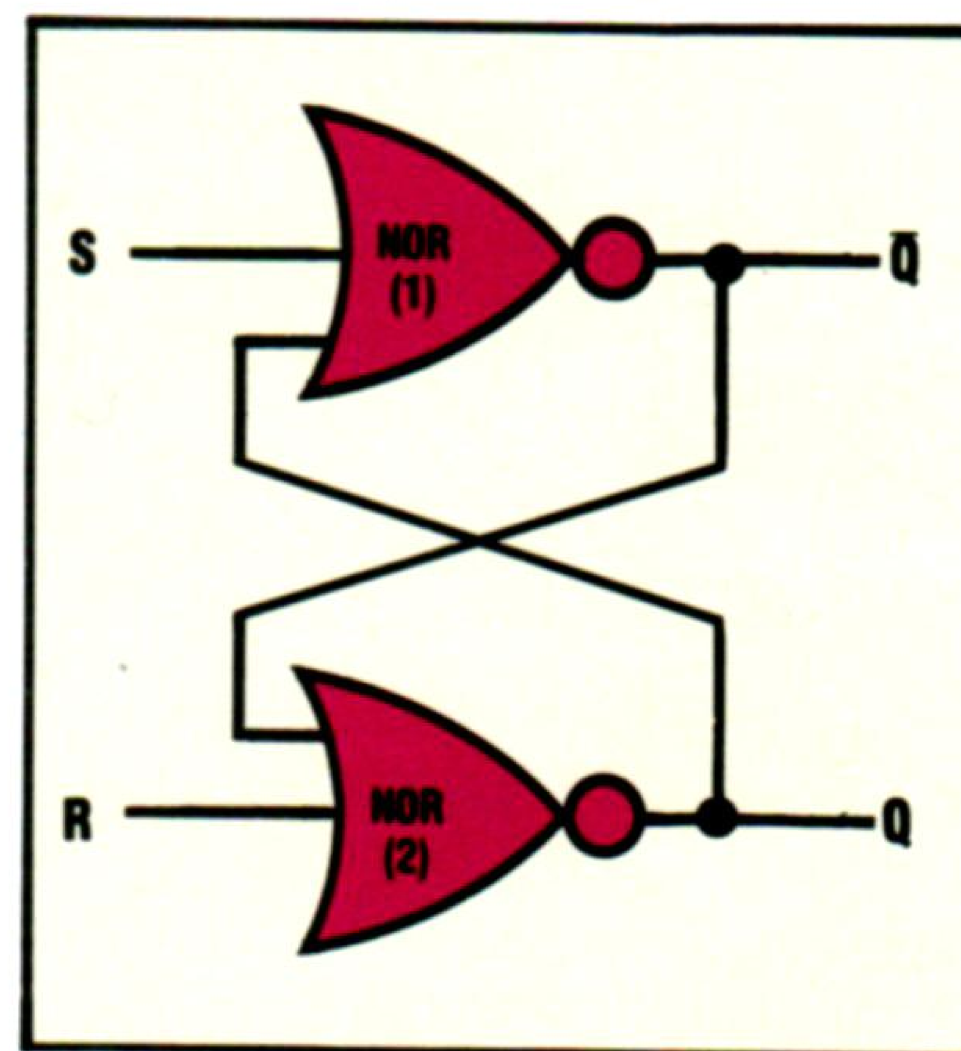
4) Un impulso en la línea RESET



Ahora, habiendo descrito la función de un biestable R-S, veamos con más detalle los elementos lógicos del circuito.

Circuito flip-flop R-S

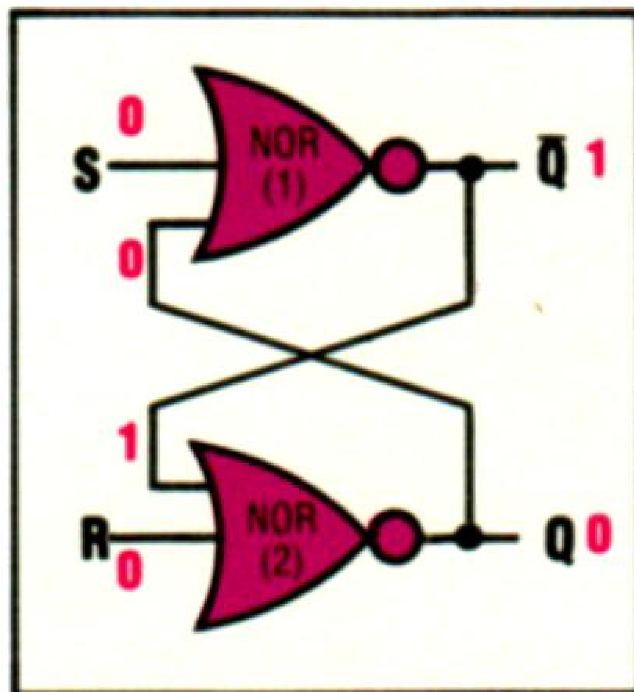
Un flip-flop R-S se puede construir utilizando varias técnicas, como enlazar entre sí dos puertas NAND o, como en el ejemplo que proporcionamos aquí, enlazando entre sí dos puertas NOR de modo tal que la salida de una de éstas constituya una de las entradas de la otra. Es este "bucle hacia atrás" de las señales lógicas lo que le proporciona al flip-flop su capacidad de "memoria".



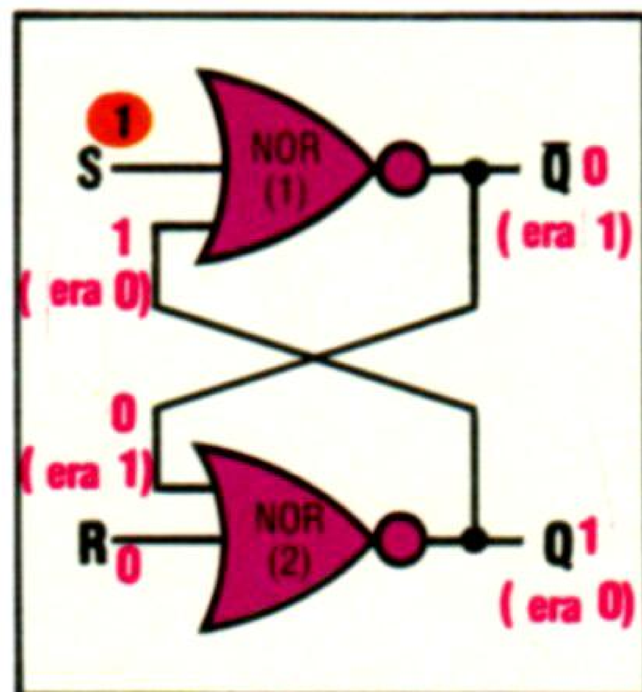
Vamos a investigar las funciones SET y RESET del flip-flop y ver cómo se obtienen mediante esta combinación de puertas NOR. Si suponemos que inicialmente el flip-flop está en estado RESET y que no hay impulsos de entrada, entonces el circuito se mantendrá en estado estable. (Recuerde que una puerta NOR sólo da una salida de uno si ambas entradas son cero.) Un impulso por la línea S modificará esta situación estable haciendo que la salida "no Q" ($\overline{Q}$) cambie a cero. Esto afecta a la entrada de la segunda puerta NOR (2), que proviene de la salida de la primera, haciendo que la salida de esa puerta, Q, cambie a uno. Esto significa que si aún está presente el impulso en la primera puerta NOR (1), entonces ambas entradas a la puerta NOR (1) serán uno. Por consiguiente, la salida de la puerta NOR (1) seguirá siendo cero y el circuito habrá cambiado al otro estado posible: SET.



1) Estado inicial (RESET)

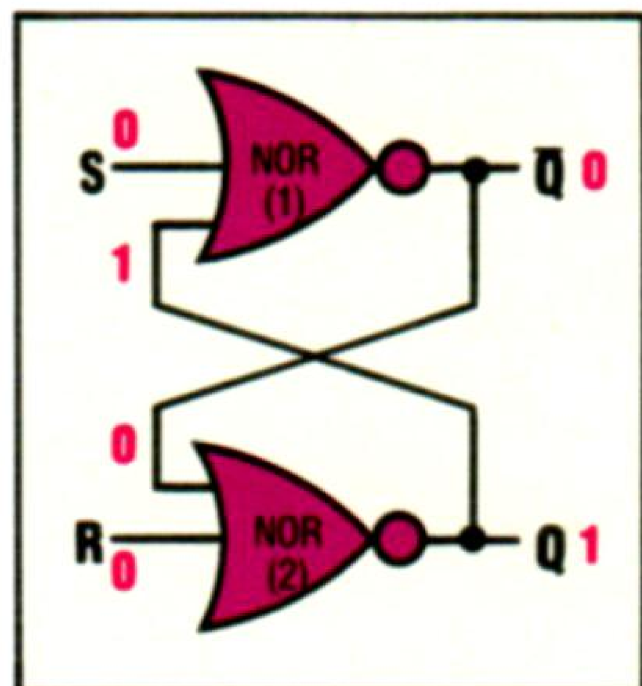


2) Un impulso en la línea SET

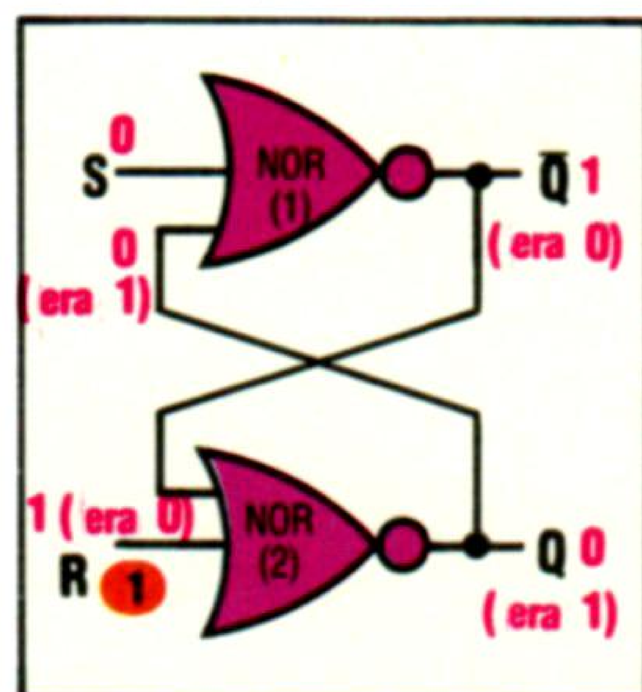


Aun cuando se elimine el impulso de la línea S, el circuito continuará estabilizado en el nuevo estado. Cuando se envía un impulso por la línea R, el circuito cambia otra vez, volviendo al estado RESET, por medio de un proceso similar al que ya hemos descrito.

3) El circuito permanece en estado SET

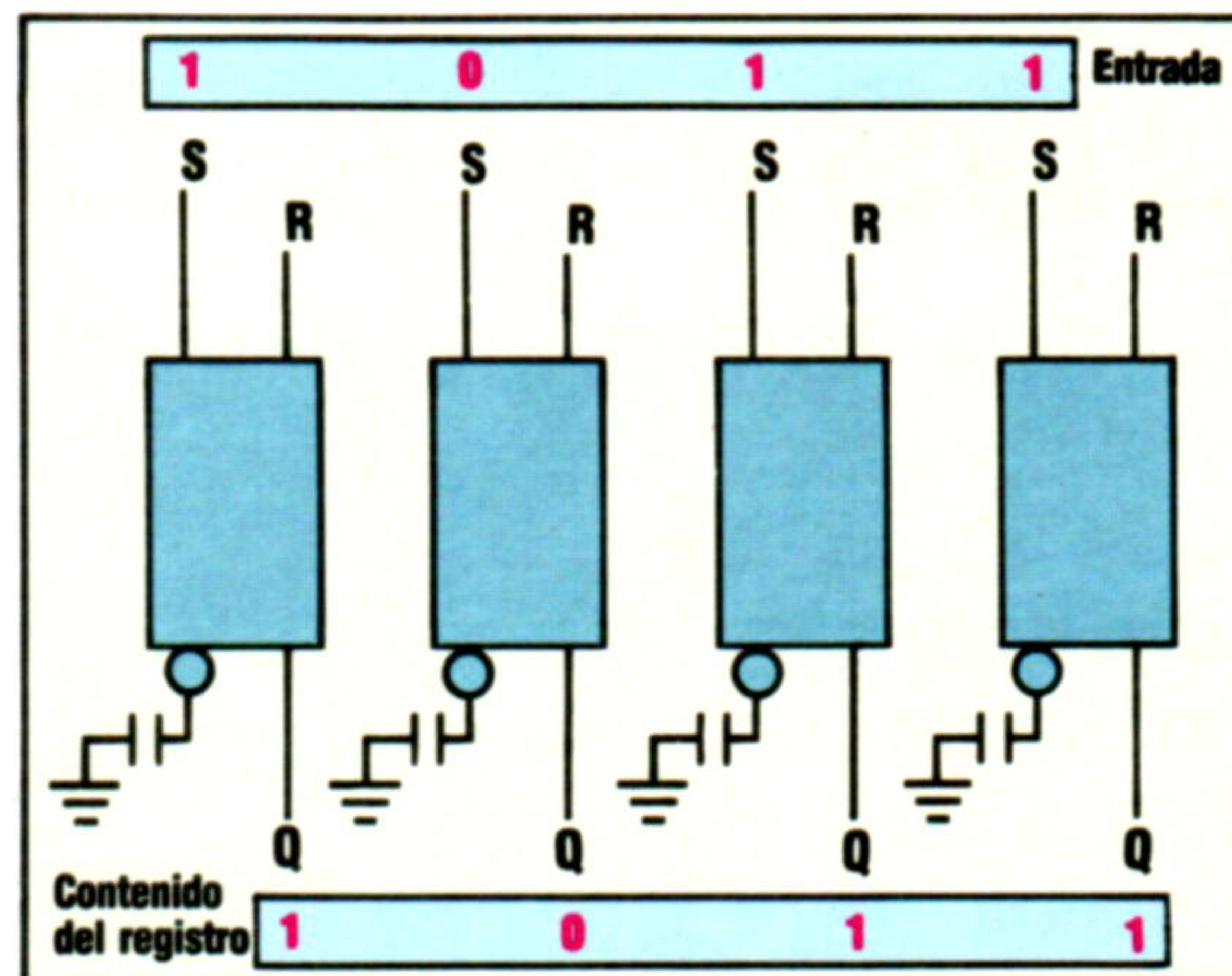


4) Un impulso en la línea RESET



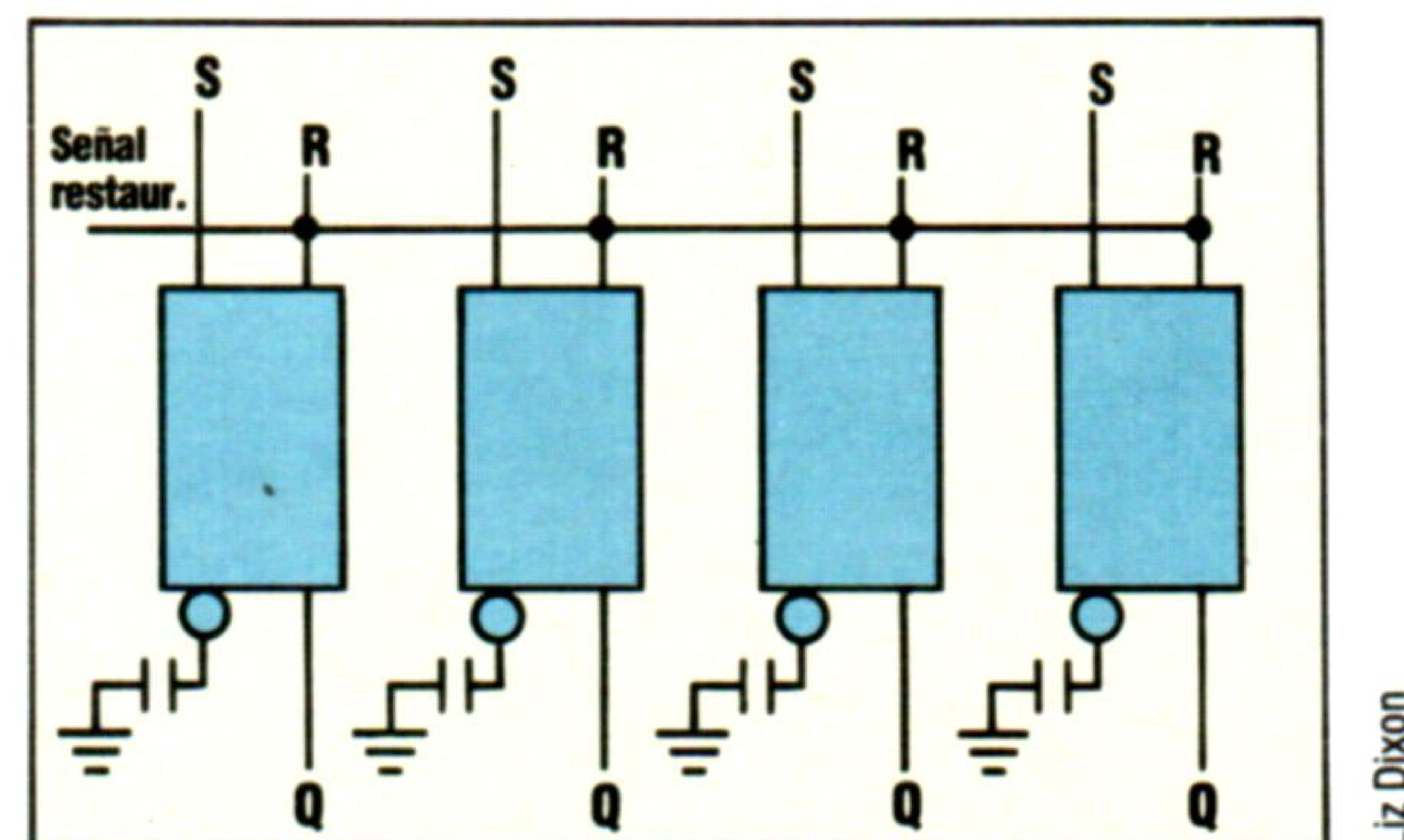
Registros

Un microprocesador se compone en gran medida de una serie de registros, como el acumulador, los registros de instrucciones y el registro índice. La mayoría de los registros pueden mantener palabras de ocho bits (es decir, números binarios comprendidos entre 0 y 255). Como estos registros tienen que aceptar y almacenar información binaria, no es sorprendente que se compongan de una serie de ocho flip-flops. Para hacer las cosas más sencillas, vamos a analizar cómo acepta y almacena números un registro de cuatro bits. Si deseamos almacenar el número binario 1011, todo cuanto se requiere es alimentar las líneas con este patrón binario.



Observe que en esta disposición la salida "no Q" no se utiliza. Como el patrón binario de entrada se aplica a las líneas S de los flip-flops, las líneas Q producen la salida correspondiente. Si deseáramos sustituir el primer número almacenado en el registro por otro, por ejemplo el 0110, podríamos pensar que todo lo que hace falta es presentar a las líneas S del flip-flop este nuevo patrón binario. De ser así, el número resultante almacenado en el registro sería 1111. Los unos de las posiciones más exteriores serían sobrantes del número anterior.

La solución para este problema consiste en restaurar cada flip-flop antes de almacenar otro número. Puesto que es necesario restaurar todos los flip-flops al mismo tiempo, es conveniente conectarlos entre sí permitiendo que la restauración del registro sea activada por una sola señal.



Liz Dixon

En el próximo capítulo analizaremos otros circuitos secuenciales, incluyendo el flip-flop tipo D y el flip-flop J-K.

Ejercicio 7

1) ¿Por qué al flip-flop también se le denomina "biestable?

2) Inicialmente, cuando se conecta un ordenador, un flip-flop está en el siguiente estado:

$Q = 0, \overline{Q} = 0, S = 0, R = 0$

a) ¿Es éste un estado estable?
b) Si no lo es, ¿a qué estado cambiará el flip-flop?
c) ¿Puede el flip-flop cambiar a un estado distinto al estado dado en su última respuesta? (Una pista: pruebe empezando con la otra puerta.)
d) ¿Qué proceso es necesario, cuando se conecta un ordenador, para asegurar que todos los registros estén en un estado predecible?

Disparos de luz

El SLR está diseñado para dar un realismo adicional a los juegos de tiro al blanco para ordenadores personales

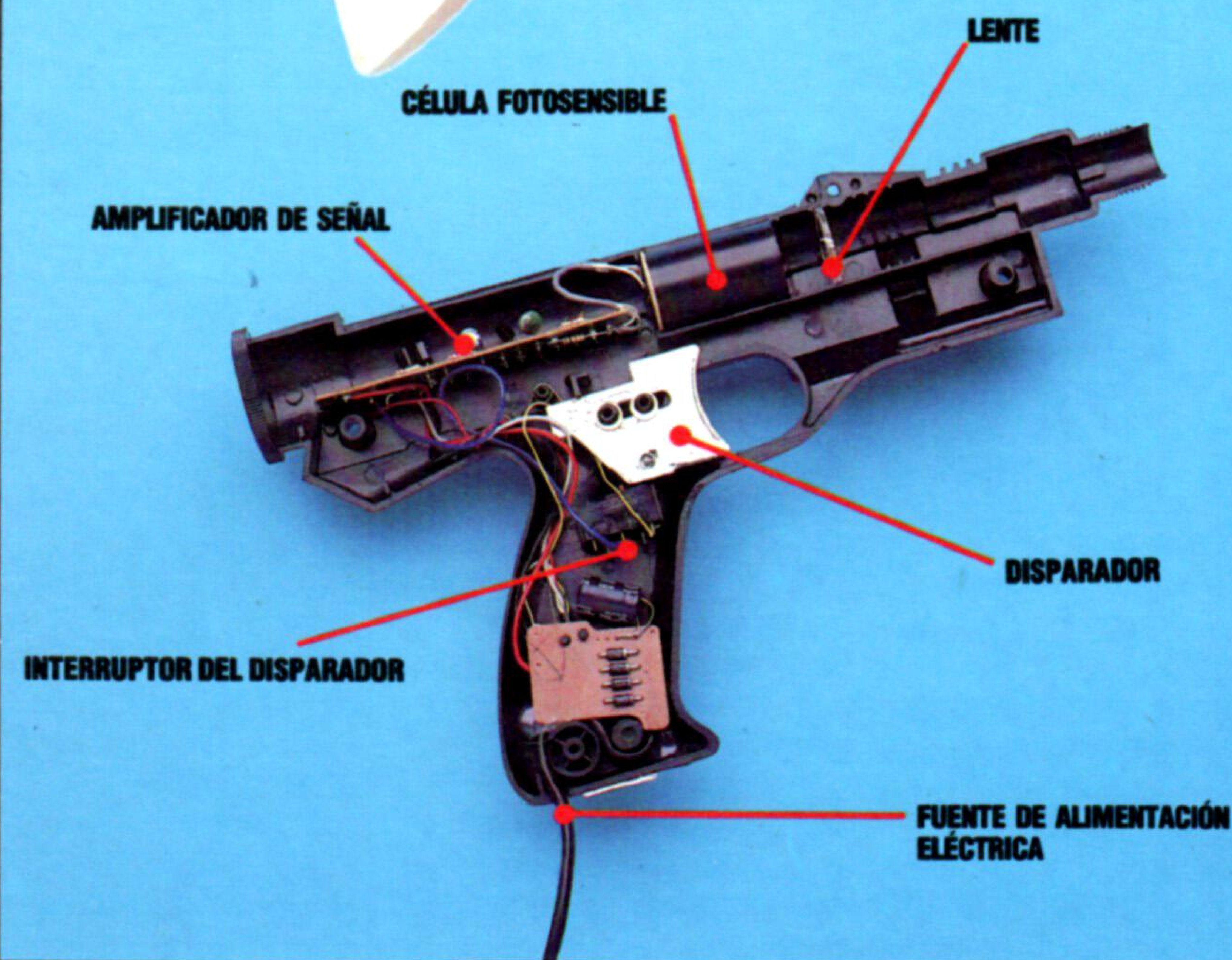
El Stack Light Rifle es un novedoso dispositivo que combina la apariencia de un fusil con un sistema óptico estilo cámara y que se basa en la tecnología del lápiz óptico (véanse pp. 156-157).

El principal componente del sistema Stack Light Rifle es la pistola electrónica receptora, que está conectada al ordenador mediante una generosa porción de cable. En un extremo del ordenador, según la versión de que se trate, hay un "enchufe" para el conector apropiado. En la versión ZX Spectrum el conector contiene dos chips y un par de componentes simples para conectar en interface la electrónica principal del interior del fusil al ordenador. Para hacer que la pistola sea más precisa (y para convertirla en rifle) se proporciona con un apoyo para el hombro que se engancha y se asegura a la culata de la pistola, además de un cañón y una mira telescópica falsa.

La electrónica del interior de la pistola se compone de un detector de luz o fotodiodo y un pequeño amplificador y buffer. La luz que penetra por el cañón se enfoca mediante una pequeña lente plástica en el fotodiodo y el dispositivo es suficientemente sensible como para detectar los cambios en la intensidad de la imagen. Una vez intensificada por el amplificador, la señal analógica se recorta para proporcionar un impulso digital, y luego se le envía al ordenador a través del interruptor. La posición de la pantalla que se está explorando en ese momento es la posición hacia la cual está apuntando el fusil. A medida que el ordenador recibe el impulso proveniente del Light Rifle, compara el valor de sus registros de exploración con la posición en pantalla del blanco y, si se halla una pareja, la jugada ha obtenido un golpe directo.

Existen variantes del Light Rifle para el ZX Spectrum, el Commodore Vic-20 y el Commodore 64 y todas ellas realizan la misma función. Stack proporciona tres juegos en cassette con el Light Rifle, y prácticamente son los únicos de que disponen. A pesar de que varias casas de software independientes producen juegos que parecerían ser eminentemente aptos para este tipo de control del usuario, muy pocas han producido o convertido programas para funcionar con él; Micromania constituye una excepción. Posiblemente aún más perjudicial para el potencial de ventas del Light Rifle es el hecho de que Stack no proporciona ninguna rutina de apoyo para permitir que los usuarios escriban sus programas. Esto, junto con la falta de detalles acerca de cómo funciona el fusil, descarta al SLR





Chris Stevens





en cuanto una posible buena alternativa a la palanca de mando.

El Light Rifle se basa en el mismo principio de operación del lápiz óptico, pero es mucho más grande y está diseñado para ser sostenido a una distancia de aproximadamente tres metros del aparato de televisión y no en contacto con la pantalla. Para ayudar a filtrar la luz ambiental, el Light Rifle dispone de un largo tubo oscuro (el cañón) y una lente. Ambos se combinan para proporcionar un grado de precisión razonable (si no perfecto) y permitir que el usuario "dispare" cómodamente desde un sofá. Los juegos que se suministran son ejemplos bastante pobres de lo que sería posible hacer; tanto el uso de gráficos como su potencial lúdico son apenas explotados.

Uno de los principales problemas de la programación para lápices ópticos, o para versiones gigantes como el Light Rifle, es que el programa ha de ser escrito de forma que sea muy eficiente. En todos los ejemplos suministrados por Stack, los juegos se interrumpen momentáneamente al apretar el gatillo. Lo ideal sería explorar continuamente la pantalla, como se realiza por lo general con un lápiz óptico, pero esto reduciría demasiado la velocidad del juego. De modo que cuando se aprieta el gatillo del Light Rifle, el software debe congelar la acción y determinar si el blanco de la pantalla está alineado con la posición de la pistola. Una vez que el software ha determinado si el jugador ha dado o no en el blanco, el juego puede continuar. En teoría, cuando se aprieta el gatillo, la cantidad de código necesario para determinar la posición en pantalla del objeto detectado por la pistola debería ser pequeña, pero ello no siempre es así.

En un ordenador como el BBC, para el cual aún no hay ninguna versión del Light Rifle, el disponer de la facilidad para lápiz óptico en el chip de video simplificaría en gran medida la tarea del software. El Commodore 64 ofrece esta clase de sistema, pero el ZX Spectrum, en el cual se probó al Light Rifle, carece de esta facilidad, y esto se refleja en el tiempo consumido en calcular la posición del rifle cuando se aprieta el gatillo.

Disparo en la oscuridad



La célula fotosensible del Light Rifle detecta el punto en movimiento de la exploración de barrido a medida que ésta refresca la imagen de televisión. El software controla continuamente el contador del raster en el ordenador, de modo que siempre conoce la posición del punto de barrido en la pantalla; cuando el fusil de luz señala que ha detectado ese punto, el software puede convertir el valor del contador del punto de barrido en las coordenadas X,Y adonde apunta el rifle

Blanco móvil



High noon (A pleno día) es el mejor de los tres programas de demostración. La animación es buena y el tirador dispara de forma verosímil. Tanto *Grouse shoot* (Caza de patos) como *Shooting gallery* (Salón de tiro) ofrecen un único blanco móvil, que ha de ser detectado antes de que salga de los límites. La animación es inestable y al apretar el gatillo se produce una perceptible interrupción en la acción en pantalla

Ian McKinnell

Crear caracteres

En este capítulo introduciremos las técnicas de los gráficos definidos por el usuario del Commodore 64 y continuaremos con el juego del "cazador de submarinos"

El juego de caracteres gráficos del Commodore 64 es de gran amplitud, pero, por lo general, es necesario crear algunos caracteres especiales o incluso volver a definir todo el juego de caracteres.

El proceso de crear sus propios caracteres en el Commodore 64 no es directo: en el BASIC Commodore no hay instrucciones para fines especiales, de modo que toda la operación se debe llevar a cabo utilizando PEEK o POKE para acceder y cambiar el contenido de la memoria.

El juego de caracteres del Commodore 64 se compone de un bloque de ROM que comienza en la posición de memoria 53248. Cada carácter aparece en la pantalla como un patrón de puntos en una matriz de puntos de ocho por ocho: describir este patrón de 64 puntos requiere 64 bits u ocho bytes. Los ocho bytes desde la posición 53248 a la 53255 describen el carácter "@", el primer carácter del juego; posee un código de pantalla 0, lo que significa que si usted coloca (POKE) el valor cero en uno de los bytes de la RAM de video, aparecerá este carácter en la pantalla.

Los ocho bytes siguientes, del 53256 al 53263, representan el carácter "A" (código de pantalla 1), y así sucesivamente.

Nosotros no podemos cambiar estas definiciones de la matriz de puntos en ROM, de modo que debemos copiar algunas de ellas o todas en RAM y hacer los cambios allí. Entonces podemos hacer que el Commodore utilice nuestro juego de caracteres de RAM para escribir en la pantalla, en vez de emplear sus propias definiciones en ROM.

El juego de caracteres en ROM comparte su espacio de direcciones en la memoria con dispositivos de entrada-salida tales como grabadoras de cassette y unidades de disco. La CPU 6510A suele tratar este espacio de memoria como una zona para entrada-salida, pero se la puede programar para que la considere como la situación del juego de caracteres. Esto podría parecer extraño, pero la CPU normalmente no hace el trabajo de acceder a las definiciones de caracteres desde ROM y enviarlas a la pantalla. Esta tarea se delega a un chip subsidiario bajo el control de la CPU. El contenido de la posición 1 determina el status de las operaciones de E/S, y el bit 2 de esta posición actúa como un indicador de la forma en que la CPU considera el juego de caracteres en ROM. Si este bit es puesto a 0, entonces la CPU asume que los dispositivos de E/S ocupan este espacio.

Los otros bits de la posición 1 poseen funciones especiales similares en el control del sistema, de modo que debemos ser cuidadosos y no alterar ninguno de ellos mientras cambiamos el valor del bit 2. La mejor forma de hacerlo es empleando los operadores lógicos AND y OR.

Supongamos que el contenido de la posición 1 es:

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	1	1	1	1

Nosotros deseamos cambiar el bit 2 a cero. Una forma de hacerlo sería calcular el valor decimal de 01101011 y colocarlo (POKE) en la posición 1, pero esto sólo funciona si sabemos que el contenido previo de la posición 1 era 01101111. Una forma mejor de modificar el bit 2 consiste en utilizar AND y PEEK. La siguiente orden coge (PEEK) el contenido de la posición 1, le aplica AND con el valor 251, y coloca (POKE) el resultado de nuevo en la posición 1:

```
POKE 1, PEEK (1) AND 251
```

El efecto de esta instrucción se ilustra así:

Bit	7	6	5	4	3	2	1	0	
	0	1	1	0	1	1	1	1	= contenido inicial
AND	1	1	1	1	1	0	1	1	= 251 binario
	0	1	1	0	1	0	1	1	= resultado de aplicar el operador AND a cada par de bits

Independientemente del valor original del bit 2, al operarlo mediante AND con cero siempre producirá un resultado cero; operando mediante AND todos los otros bits de la posición con uno simplemente mantiene su valor original. El número binario 11111011 (decimal 251) se denomina *máscara* u *overlay*, y aquí lo estamos empleando como una "máscara AND".

Para poner el bit 2 a uno sin afectar a ninguno de los otros bits, utilizamos la siguiente orden:

```
POKE 1, PEEK (1) OR 4
```

Bit	7	6	5	4	3	2	1	0	
	0	1	1	0	1	0	1	1	= contenido inicial
OR	0	0	0	0	0	1	0	0	= 4 binario
	0	1	1	0	1	1	1	1	= resultado de aplicar el operador OR a cada par de bits

Esto asegura que el BASIC no machacará nuestro juego de caracteres. Cuando la copia está completa, la CPU se puede restaurar para direccionar los dispositivos de E/S restaurando la interrupción.

La pieza final de este rompecabezas es obligar al chip de manipulación de pantalla a que utilice nuestro juego de caracteres en vez del juego de la ROM del sistema. Los bits del 0 al 3 de la posición 53272 señalan la dirección de inicio del juego de caracteres, y la tabla siguiente muestra cómo el Commodore 64 interpreta que los valores de estos bits apuntan a direcciones determinadas:

valor decimal de bits 0 al 3	bits 3, 2, 1, 0	posición a que apuntan
0	0000	0
2	0010	2048
4	0100	4096
6	0110	6144
8	1000	8192
10	1010	10240
12	1100	12288
14	1110	14336

En este registro el valor del bit 0 es irrelevante, mientras que los bits del 4 al 7 controlan otras funciones y no deben ser alterados. Con este fin, empleamos 11110000 (decimal 240) como una máscara AND, y 00001110 (decimal 14) como una máscara OR para hacer que el registro apunte a 14336, la posición inicial de nuestro juego de caracteres:

```
POKE 53272, (PEEK (53272) AND 240) OR 14
```

Ahora, utilizando un bucle FOR...NEXT podemos realizar la copia en curso.

Mientras un programa está copiando en RAM el juego de caracteres de ROM, la CPU no puede tratar interrupciones de dispositivos de E/S. El teclado, por ejemplo, interrumpe a la CPU cada sesentavo de segundo, haciéndola explorar el teclado en busca de alguna pulsación. Estas interrupciones son provocadas por el reloj del sistema. Si un dispositivo de E/S interrumpiera a la CPU mientras el juego de caracteres está ocupando el espacio de ROM para E/S (como en este caso durante el copiado), entonces probablemente el sistema se colgaría y la máquina sólo podría ser reinicializada desconectándola y volviéndola a conectar a la corriente. Por suerte, podemos bloquear el mecanismo de interrupción poniendo a cero el bit 0 de la posición 56334; los otros bits de esta posición no se deben modificar, de modo que hay que utilizar la siguiente instrucción lógica POKE:

```
POKE 56334, PEEK (56334) AND 254
```

Una vez bloqueadas las interrupciones y obligada la CPU a mirar el juego de caracteres en ROM, entonces se puede iniciar la copia.

Nosotros queremos copiar, pongamos por caso, los 64 caracteres desde "@" a "?" (códigos de pantalla de 0 a 63), de modo que debemos copiar las 512 posiciones (8×64=512) comenzando desde 53248 en un bloque de RAM adecuado. Se pueden utilizar varias zonas, y aquí nuestra elección es un bloque que comienza en la posición 14336. Ésta normalmente estaría en el área para programas BASIC, pero nosotros la protegemos bajando el puntero a la parte superior de la memoria que hay en la posición 56, de este modo:

```
POKE 56,32
```

Cada carácter del juego del Commodore está diseñado sobre una matriz de puntos de ocho por ocho. Cada fila de la matriz se interpreta como un número binario (los puntos que están iluminados se interpretan como uno, los que no aparecen en la pantalla se interpretan como cero) y requiere un byte de memoria, y las ocho filas del carácter completo requieren ocho posiciones consecutivas en la memoria. La posición inicial de los bytes que describen un carácter se puede calcular a partir de la dirección inicial del bloque entero y el código de pantalla del carácter en cuestión, de este modo:

Comienzo de carácter = 14336 +8× (código de pantalla)

Una vez que se conocen las posiciones de los bytes que definen a un carácter, podemos colocar (POKE) nuevos valores en estos bytes cambiando, por consiguiente, los patrones de puntos que aparecen en la pantalla cuando se visualiza el carácter. Siempre que el juego de caracteres está activado, pulsar la tecla correspondiente hará que el nuevo carácter aparezca en la pantalla. En el programa de demostración, redefinimos los caracteres [,], £ y ↑ (códigos 27-30) como partes de una figura, y conseguimos su animación imprimiendo y sobreimprimiendo distintas versiones de la figura.

```
130 :
140 REM **** COPIAR JUEGO CARACT. ROM ****
150 POKE56,32
    :REM BAJAR TOPE SUPERIOR MEMORIA
160 POKE56334,PEEK(56334)AND254
    :REM DESCONECTAR RELOJ INTERRUPCIONES
165 POKE1,PEEK(1)AND251
    :PERMUTAR A CARACT. ROM
170 FOR I=0 TO 511       :REM COPIAR
180 POKE14336+I,PEEK(53248+I)
    :REM 64
190 NEXT I
    :REM CARACTERES
200 POKE1,PEEK(1)OR4
    :PERMUTAR A E/S
210 POKE56334,PEEK(56334)OR1
    :REM CONECTAR RELOJ INTERRUPCIONES
220 POKE53272,(PEEK(53272)AND240)OR14:    REM
    PONER PUNTERO CARACT.
230 REM **** COPIA COMPLETA ****
240 :
250 :
300 REM **** EL GIMNASTA ARTURO ****
310 FOR I=14552TO14552+31
    :REM LEER
320 READ A:POKEI,A: NEXT I
    :REM DATOS CARACT.
330 PRINTCHR$(147)
    :REM BORRAR PANTALLA
340 POKE55338,14:POKE55378,14
    :CARACT. COLOR CELESTE
350 POKE1066,27:POKE1106,28
    :REM BRAZOS ARRIBA
360 FORI=1TO500:NEXT I
    :REM BUCLE DEMORA
370 POKE1066,29:POKE1106,30
   :REM BRAZOS ABAJO
380 FORI=1TO500: NEXT I
    :REM BUCLE DEMORA
390 GOTO350
480 :
490 :
500 REM **** DATOS BRAZOS ARRIBA ****
510 DATA129,153,189,153,66,60,60,60
520 DATA60,60,36,36,66,66,195,0
530 REM *** DATOS BRAZOS ABAJO ****
540 DATA0,24,60,24,0,126,189,189
550 DATA189,189,36,36,36,36,102,0
```

128	64	32	16	8	4	2	U	DECIMAL
1	0	0	0	0	0	0	1	129
1	0	0	1	1	0	0	1	153
1	0	1	1	1	1	0	1	189
1	0	0	1	1	0	0	1	153
0	1	0	0	0	0	1	0	66
0	0	1	1	1	1	0	0	60
0	0	1	1	1	1	0	0	60
0	0	1	1	1	1	0	0	60

0	0	1	1	1	1	0	0	60
0	0	1	1	1	1	0	0	60
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	1	0	0	0	0	1	0	66
0	1	0	0	0	0	1	0	66
1	1	0	0	0	0	1	1	195
0	0	0	0	0	0	0	0	0

128	64	32	16	8	4	2	U	DECIMAL
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	24
0	0	1	1	1	1	0	0	60
0	0	0	1	1	0	0	0	24
0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	126
1	0	1	1	1	1	0	1	189
1	0	1	1	1	1	0	1	189

1	0	1	1	1	1	0	1	189
1	0	1	1	1	1	0	1	189
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	1	1	0	0	1	1	0	102
0	0	0	0	0	0	0	0	0

Liz Dixon

Aspecto de la figura
Los caracteres se construyen en una matriz de puntos de ocho por ocho, descrita en ocho bytes consecutivos. Cada fila del carácter se interpreta como un número binario de un único byte, representando los puntos por 1 y los espacios por 0. Para utilizarlos en programas para el Commodore 64, estos números binarios se deben convertir a decimal

Trazando el Subhunter

Éste es el trazado para el juego Subhunter del Commodore mostrado en forma de diagrama de flujo. Hasta ahora hemos inicializado el programa y hemos puesto el reloj en cero. El siguiente paso será dar entrada al bucle del programa inicial, que hará que el barco y el submarino se desplacen mientras se actualizan el marcador y el reloj



INICIO
Establecer pantalla Inicializar marcadores
Leer datos sprites Inicializar sprites
Seleccionar velocidad y profundidad del submarino
Poner reloj en cero
Actualizar reloj
¿Es el reloj > 2.59? S / N
Apagar sprites Cambiar posición barco
Seleccionar velocidad y profundidad del submarino
¿Otro juego? S / N
Actualizar marcador máx.
Restaurar marcador
Restaurar reloj
Encender barco y submarino
FINAL
Aceptar entrada del teclado
Mover barco y submarino
¿Submarino en borde pantalla? S / N
Actualizar marcador
Seleccionar velocidad y profundidad del submarino
¿Se ha lanzado una carga? S / N
Mover carga profundidad
¿Hay una colisión? S / N
Actualizar marcador
Seleccionar velocidad y profundidad del submarino

Programa Subhunter

El Commodore 64 posee su propio reloj interno que se puede utilizar para sincronizar programas en BASIC. El reloj tiene seis dígitos, parecidos a un reloj digital, que representan horas (00-23), minutos (00-59) y segundos (00-59). El reloj puede ser accedido desde BASIC con la variable TI$. El valor de TI$ da el tiempo transcurrido desde que se conectó el ordenador a la red, pero se puede reinicializar en cualquier momento. El siguiente programa muestra cómo funciona el reloj:

```
10 REM **** RELOJ ****
20 PRINT CHR$(147)      : REM BORRAR
                          PANTALLA
30 TI$ = "000000"       : REM PONER RELOJ
                          EN CERO
40 PRINTCHR$(145);TI$:  REM IMPRIMIR
                          VALOR EN CURSO
                          DEL RELOJ
50                      : REM CHR$(145)=
                          CURSOR ARRIBA
60 GOTO40
```

El programa se ejecuta en un bucle continuo, visualizando el reloj en la pantalla hasta que se pulse la tecla Run/Stop. El juego que estamos escribiendo exige la visualización de un reloj en la pantalla y darlo por terminado después de transcurridos tres minutos. El reloj sólo requiere las partes de TI$ correspondientes a los minutos y segundos. Utilizando las funciones string podemos dividir TI$ así:

RIGHT$(TI$,2)

TI$=HH(MM)(SS)

MID$(TI$,3,2)

Los dos dígitos de los segundos se pueden aislar mediante RIGHT$ (TI$,2) y los dígitos de los minutos se pueden aislar mediante MID$(TI$,3,2).

El bucle principal del programa empieza en la línea 200 y termina en la 390. Cargue la subrutina descrita en la última sección y añada estas líneas:

```
140 TI$="000000"
150 :
160 :
200 REM **** BUCLE PRINCIPAL ****
205 :
210 REM ** RELOJ **
220 PRINTCHR$(19);:TAB(14)CHR$(5)"TIEMPO
    ";MID$(TI$,3,2);":"; RIGHT$(TI$,2)
225 IFVAL(TI$)>259THEN400: REM FINAL JUEGO
:
390 GOTO200: REM RECOMENZAR BUCLE
    PRINCIPAL
400 END
```

La línea 140 restablece el reloj en la posición inicial del programa. La línea 220 imprime (PRINT) el actual valor del reloj en minutos y segundos, separados por un punto y coma. TAB (14) hace que queden libres 14 espacios antes de imprimir y posiciona el reloj en el medio de la pantalla. CHR$(5) dará color blanco a los caracteres. La línea 225 transforma TI$ en cantidad numérica. Si el tiempo de juego ha excedido los 2 minutos 59 segundos, éste ha llegado a su fin.

Del diagrama al BASIC

Los símbolos de los ordinogramas tienen una perfecta transcripción a instrucciones BASIC

Todos aquellos símbolos que hasta ahora se han asociado a ideas pueden asimilarse a instrucciones determinadas. Así, el símbolo de entrada por teclado se codificará con el mando INPUT, y el rectángulo, símbolo de operación, lo será como LET, si bien la mayoría de sistemas prescinden de su utilización, representándose su contenido en forma algebraica: A = B equivale a LET A = B.

El rombo (símbolo de decisión) se traduce por IF. La visualización o la impresión, según se emplee o, se transcribe por PRINT, mientras que las líneas de flujo y los conectores se representan por GOTO. En cuanto a los terminales, el final equivale a END (en el Spectrum, STOP), y el inicial, y a modo de título, puede representarse con un REM, que también servirá para incluir posibles comentarios. Veamos un ejemplo.

El siguiente ejemplo plantea la visualización de los números pares comprendidos entre 2 y 50, así como sus cuadrados (fig. 1). Se ha utilizado A para denominar al contador y C para el cuadrado. Las líneas 60 y 70, que forman la decisión, pueden transcribirse así:

```
60 IF A < 50 OR A = 50 THEN GOTO 30
70 END
```

o bien

```
60 IF A > 50 THEN END
70 GOTO 30
```

Modifiquemos la situación, suponiendo que en lugar de dos números determinados se entren por teclado dos cifras cualesquiera (fig. 2). Se observa entonces que la primera cantidad debe ser siempre par, para que el incremento de dos en dos devuelva el resultado deseado. Pero dicha tarea no queda encomendada a la persona que dé entrada a la cantidad que desee. Debe ser por programa como se controle dicho requisito. Asimismo, debe comprobarse que, dado que el contador se incrementa, la primera cantidad debe ser menor que la segunda, caso contrario se invertirá el orden de las cantidades, con lo cual se visualizarán siempre los números pares de menor a mayor.

```
10 REM PARES Y CUADRADOS
20 INPUT "PRIMERA CANTIDAD";A
30 INPUT "SEGUNDA CANTIDAD";B
35 REM CONTROL DE CANTIDADES
40 IF A<B THEN GOTO 60
50 X=B: B=A: A=X
55 REM CONTROL PARIDAD DE A
60 IF A/2=INT (A/2) THEN GOTO 80
70 A=A+1
75 REM INICIO DEL BUCLE
80 C=A+A
90 PRINT A,C
100 A=A+2
110 IF A>B THEN END
120 GOTO 80
```

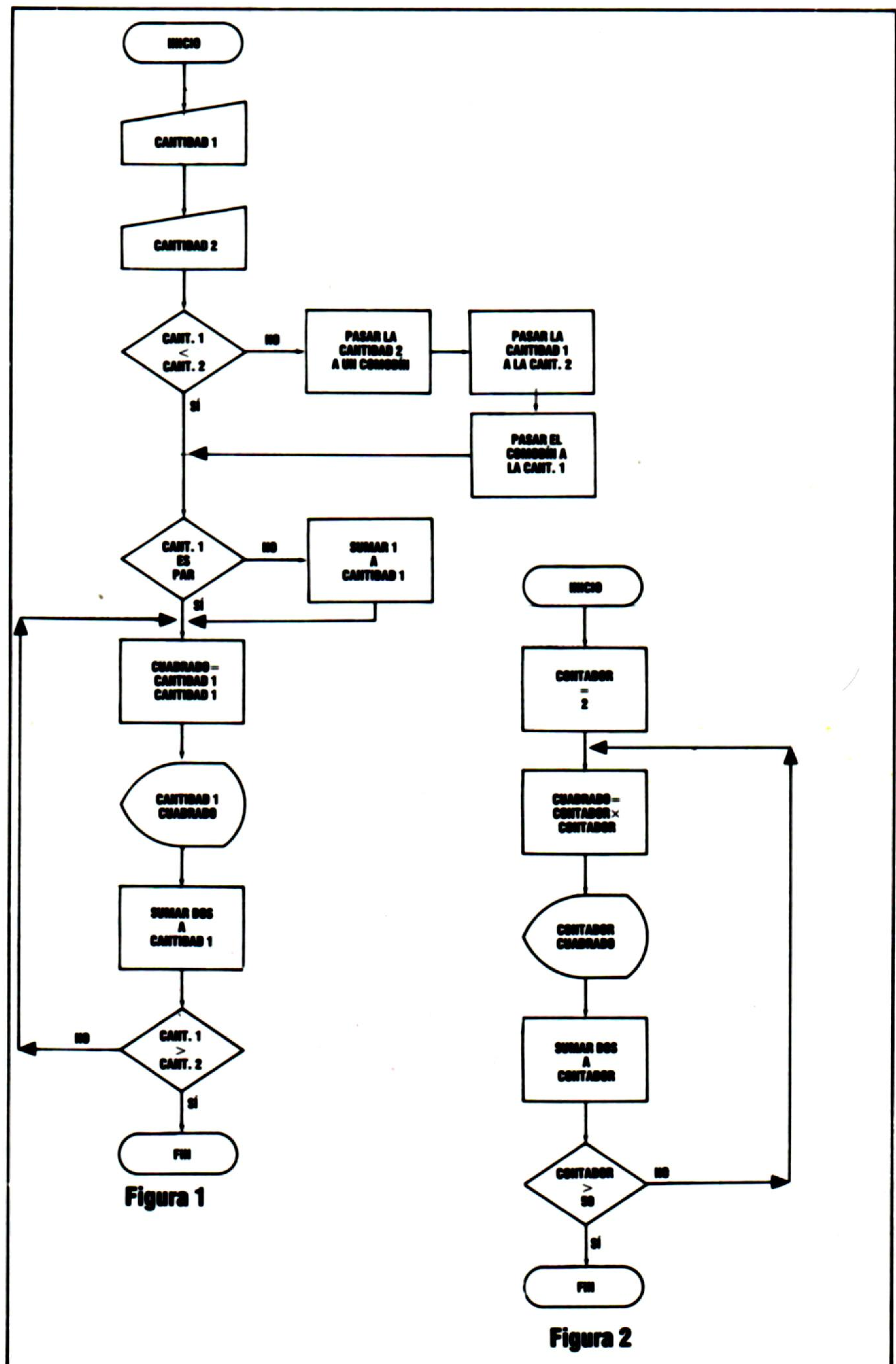


Figura 1

Figura 2

Rutina general

Estudiamos los símbolos y etiquetas que hacen manejable el lenguaje assembly y los aplicamos a algunas subrutinas

Debido a que el lenguaje assembly es esencialmente un lenguaje de programación simple que se compone de órdenes "primitivas" que pueda administrar la CPU, usted se encontrará escribiendo y reescribiendo fragmentos de programa para realizar las mismas tareas esenciales que forman parte de las instrucciones de un lenguaje de alto nivel: cómo tratar la entrada-salida, por ejemplo, o las rutinas aritméticas con dos bytes. Lo más recomendable es formar una biblioteca (en cinta, en disco o en papel) de las rutinas que se utilizan con mayor frecuencia para incluirlas en bloque en los nuevos programas en el momento en que se plantee la necesidad de emplearlas.

Existen a este respecto dos problemas fundamentales.

El primero reside en la dificultad de escribir rutinas importantes y, a veces, largas, de una forma que resulte lo suficientemente general como para poder insertarlas en programas diferentes sin reajustes ni reescritura.

El segundo problema está en escribir rutinas útiles que no dependan de unas determinadas posiciones de memoria, y se puedan volver a posicionar en la memoria por medio de un nuevo ensamblaje con una dirección ORG distinta para desempeñar aquí exactamente la misma función que en su posición original.

Ambos problemas son aspectos de un solo tema, el de la generalidad-adaptabilidad, conocido por los programadores de BASIC, y se resuelven de forma bastante similar: hay que recurrir a variables para pasar los valores del programa a la subrutina; usar variables locales en las subrutinas para hacerlas independientes del contexto del programa principal; y, finalmente, evitar de lo posible la utilización de variables absolutas (constantes tanto numéricas como en serie) así como los números de línea de programa.

En la programación en lenguaje assembly nos hemos habituado a la idea de las posiciones de memoria como el equivalente de las variables del BASIC; los programas operan sobre el contenido de estas posiciones, cualesquiera que sean dichos contenidos, igual que un programa en BASIC opera sobre el contenido de sus variables. Lamentablemente, hemos tendido a referirnos a las posiciones de memoria por sus direcciones absolutas, un hábito que al principio era conveniente pero al que en este momento debemos renunciar en favor de la generalización.

La respuesta consiste en utilizar símbolos en lugar de direcciones absolutas y valores, por una parte; y, por otra, en recurrir a la utilización de la gama de formas simbólicas que ofrecen los pseudo-opcodes para el ensamblador como los equivalentes tanto de las variables como de los números de línea de programas. Ya hemos visto anteriormente, en otros apartados del presente curso avanzado, algunos ejemplos de esto.

Habiendo llegado a este punto, consideremos el siguiente programa:

6502			Z80		
DATA1	EQU	$12	DATA1	EQU	$12
DATA2	EQU	$79	DATA2	EQU	$79
	LDA	DATA1		LD	A,(DATA1)
LOOP	ADC	DATA2		ADC	A,(DATA2)
	BNE	LOOP		JR	NZ,LOOP
	RTS			RET	

Aquí hemos utilizado dos tipos de símbolos, dos valores y una etiqueta, todos empleados como operandos de las instrucciones en lenguaje assembly. Debido a ello, el fragmento de programa es a la vez general y transportable. Las únicas cantidades absolutas son los valores de DATA1 y DATA2, y se pueden inicializar en el programa principal sin necesidad de hacerlo al comienzo de la rutina.

Hay otros pseudo-ops que no hemos analizado todavía. En especial, DB, DW y DS (si bien al igual que ORG y EQU, pueden variar de un programa ensamblador a otro). Se trata de siglas inglesas: DB por **D**efine **B**yte (definir byte), DW por **D**efine **W**ord (definir palabra) y DS por **D**efine **S**torage (definir almacenamiento), nos permiten inicializar y designar posiciones de memoria, como en este ejemplo:

			ORG	$D3A0
D3A0	5F	LABL1	DB	$5F
D3A1	CE98	LABL2	DW	$98CE
D3B3		LABL3	DS	$10
D3B3		DATA1	EQU	LABL3

TABLA DE SÍMBOLOS:
LABL1 = D3A0: LABL2 = D3A1: LABL3 = D3A3
DATA1 = D3A3
ASSEMBLY COMPLETO — NINGÚN ERROR

En este listado completo de assembly (la salida de un programa ensamblador) vemos en la parte inferior y por primera vez una tabla de símbolos, con los símbolos definidos en el programa y los valores que representan. En este fragmento hay varias cosas importantes que observar. Antes que nada, en la línea que comienza con LABL1 se utiliza el pseudo-op DB. En el listado podemos ver que la directriz ORG ha atribuido a LABL1 la dirección $D3A0, y la tabla de símbolos así lo confirma. Aquí DB tiene el efecto de colocar el valor $5F en el byte direccionado por LABL1; de modo que la posición de memoria $D3A0 contiene inicialmente el valor $5F, tal como podemos ver en la columna de lenguaje máquina del listado.

En segundo lugar, LABL2 hace las veces de la dirección $D3A1. Pero como DW ha de colocar en ella una "palabra" (o sea, dos bytes consecutivos), el valor $98CE se almacena en las posiciones $D3A1 y $D3A2 en forma *lo-hi* (esto se puede ver con toda



claridad en la columna de lenguaje máquina). Dado que DW convierte automáticamente sus operandos a forma *lo-hi*, suele utilizarse para inicializar posiciones de "punteros" con direcciones. LABL2, o la posición $D3A1, podría ser una de tales direcciones: señala la posición $98CE.

Otro punto a considerar es que la instrucción DS $10 tiene el efecto de añadir $10 al contador del programa. Esto está más claro en la tabla de símbolos que en el listado; LABL3 es la posición $D3A3 (la posición que sigue a la instrucción anterior), pero en el listado resulta que su valor es $D3B3. Esto se explica porque la dirección se refiere a la siguiente instrucción después de DS, de modo que DS $10 ha reservado un bloque de 16 bytes (de $D3A3 a $D3B2 inclusive) entre una instrucción y la siguiente. Es un procedimiento parecido al de colocar líneas REM largas en un programa en BASIC con el fin de crear espacio sin utilizar en el área para texto de programas, donde se pueda entonces colocar (POKE) y recuperar (PEEK) un área de programas en lenguaje máquina (véase p. 617).

Por último, se utiliza EQU para establecer un símbolo igual al valor de otro, de modo que DATA1 tiene el valor $D3A3 (el valor de LABL3). Aquí tenemos otra fuente de posible confusión. LABL3 es la representación simbólica de la dirección $D3A3, de manera que DATA1 EQU LABL3 significa "el símbolo DATA1 tendrá el mismo significado y valor que LABL3". El hecho de que la instrucción DB haya hecho que el contenido de $D3A3 fuera igual a $5F no tiene importancia alguna para el significado de los símbolos LABL3 y DATA1. Tener siempre presente con toda claridad la diferencia entre una dirección y su contenido es uno de los ejercicios más aconsejables en las primeras etapas de aprendizaje de la programación en assembly. Es muy probable que usted haya experimentado con anterioridad esta misma sensación con las variables y sus contenidos programando en BASIC.

A primera vista, la directriz DB parece un equivalente de EQU, pero no es así. LABL1 significa "la dirección $D3A0", y DB $5F ha inicializado ese byte con el valor $5F, pero, aunque el valor de LABL1 ahora sea fijo, el contenido de la dirección que simboliza puede cambiar en cualquier momento (p. ej., almacenando allí, según avanza el programa, el contenido del acumulador). De forma similar, ahora DATA1 es un símbolo cuyo valor está fijado por la instrucción EQU; la ejecución del programa no puede cambiar su valor. Y, nuevamente, LABL3 señala el comienzo del área de datos de 16 bytes, cuyos contenidos pueden cambiar en el programa, pero LABL3 en sí mismo es invariable.

Esto aclara, pero no agota, ni mucho menos, la amplia gama de posibilidades que los nuevos pseudo-ops ofrecen al usuario. Consideremos ahora esta nueva versión del fragmento anterior:

			ORG	$D3A0
D3A0	4D4553	LABL1	DB	'MESSAGE 1'
D3A9	CE98	LABL2	DW	$98CE
D3BB		LABL3	DS	$10
D3BB		DATA1	EQU	LABL3

TABLA DE SÍMBOLOS:
LABL1 = D3A0: LABL2 = D3A9: LABL3 = D3AB
DATA1 = D3AB
ASSEMBLY COMPLETO - NINGÚN ERROR

A la instrucción DB sigue un string, 'MESSAGE 1', como operando, y el ensamblador ha inicializado las posiciones desde $D3A0 hasta $D3A8 con los valores ASCII de los caracteres encerrados entre comillas simples.

Lo que se deduce de la columna que cuenta las direcciones de posición, y es parcialmente confirmado por la columna con el código de lenguaje máquina: los contenidos de los tres bytes desde $D3A0 hasta $D3A2 resultan ser $4D, $45 y $53, que corresponden a los valores ASCII en hexadecimal de las letras 'M', 'E' y 'S'.

Ésta es una facilidad importante, no sólo porque alivia al programador en la tarea de traducir mensajes y datos de caracteres a listas de códigos ASCII, sino también porque hace que el listado resulte mucho más fácil de leer, y sugiere la posibilidad de obtener alguna salida en pantalla con programas en lenguaje assembly. Esto último es especialmente notable, porque hasta ahora nos hemos limitado a almacenar resultados en la memoria e inspeccionarlos utilizando el programa monitor (véase p. 598).

Naturalmente, en este curso analizaremos cómo tratar la pantalla, pero existen todavía algunos aspectos del lenguaje assembly que necesitamos investigar antes de entrar en ese tema. No obstante, si usted piensa en nuestra costumbre de almacenar resultados en la memoria y si ya ha comprendido que las visualizaciones en pantalla por mapas de memoria son, en realidad, sólo áreas de la memoria, quizá ya intuye la forma de direccionar la pantalla desde un programa.

El aspecto más importante de esta nueva facili-

Ejercicios

1) El primer fragmento de programa del texto principal emplea el pseudo-op DS para reservar $10 bytes de memoria comenzando desde la dirección representada mediante la etiqueta LABL1. Escribir un programa en lenguaje assembly que almacene los números de $0F a $00 por orden descendente en este bloque, a un número por byte. Se ha de usar, para hacerlo, un bucle y técnicas de direccionamiento indexado, para lo que habrá de utilizar las instrucciones DEX (disminuir el registro X) o DEC (IX+0) (disminuir IX). El bucle continuará mientras la disminución del registro índice no ponga en 1 el flag de cero, o sea, ha de utilizar las instrucciones de bifurcación BNE o JR NZ.

2) Utilizando las técnicas del ejercicio anterior, escriba un programa para copiar el mensaje almacenado en LABL1 por el pseudo-op DB (véase el segundo fragmento de programa del texto principal) en un bloque de memoria que comience en la dirección almacenada en LABL2 por el pseudo-op DW. Puede que la dirección $98CE no sea adecuada para su ordenador, y que cambien las posiciones iniciales; pero el programa debe funcionar para cualquier dirección y para mensajes de cualquier longitud. Para implementar esto, su programa debe utilizar el número de caracteres del mensaje como un contador del bucle, o bien debe ser capaz de reconocer el final del mensaje; podría colocar un asterisco, por ejemplo, como último carácter de todo mensaje.

dad DB es que da a LABL1 el rango de una variable string de BASIC. Cuando escribimos en este lenguaje:

```
200 LET A$+"MESSAGE 1"
```

lo que hacemos es crear un señalador del comienzo de una tabla de bytes que contienen los valores ASCII de 'M', 'E', 'S', etc. Cada vez que el intérprete de BASIC encuentra una referencia a A$, busca en su propia tabla de símbolos para hallar la posición que se señala; es decir, la posición de comienzo de los contenidos de A$. Como decimos, en nuestro programa en assembly podemos tratar LABL1 como el equivalente de A$, pues ya hemos escrito un fragmento de programa que nos permite manipular una tabla utilizando el direccionamiento indexado.

Los pseudo-ops, pues, eliminan de nuestros programas direcciones absolutas y valores, y se reemplazan por símbolos. Con esto se disminuyen los problemas de generalización y adaptabilidad. Lo que necesitamos ahora es saber acceder a estos módulos portátiles y adaptables desde el programa principal.

En otras palabras, es preciso contar con un equivalente en lenguaje máquina de la instrucción GOSUB del BASIC.

Existen, por supuesto, instrucciones de estas características: JSR y CALL, en 6502 y Z80, respectivamente. Ambas emplean una dirección absoluta (que puede ser una etiqueta) como operando, y ambas tienen el efecto de sustituir los contenidos del contador del programa con la dirección que constituye su operando. La siguiente instrucción a ejecutar, por lo tanto, será la primera instrucción de la subrutina así direccionada. La ejecución prosigue a partir de esa instrucción hasta encontrar la instrucción RETURN (RTS y RET, respectivamente). Esta orden tiene el efecto de reemplazar los contenidos corrientes del contador del programa con los contenidos inmediatamente anteriores a la ejecución de la instrucción JSR o CALL. La siguiente instrucción a ejecutar, por lo tanto, es la que sigue inmediatamente a JSR o CALL. Éste es exactamente el mecanismo de que se vale el intérprete de BASIC para ir y venir de las sentencias aludidas por GOSUB. Como tal, resulta fácilmente comprensible, pero plantea la cuestión de cómo se restauran los antiguos contenidos del contador del programa una vez ejecutada la instrucción RETURN. Es fácil entenderlo: las instrucciones JSR y CALL primero "empujan" los contenidos del contador del programa en la pila (véase ilustración de p. 616) antes de reemplazarlos por la dirección de la subrutina; y las instrucciones RTS y RET "hacen saltar" esa misma dirección fuera de la pila para devolverla al contador del programa.

Qué es la pila, cómo se "empuja" o se hace "saltar" y por qué uno ha de hacerlo constituyen los temas del próximo capítulo del curso.

Juego de instrucciones

BEQ — BIFURCAR EN CERO — 6502

Relativo F0 (2 bytes)

El valor del byte que sigue al opcode desplaza el contenido del contador del programa.

EFECTO EN EL PSR

S V B D I Z C

MSB [| | | | | | |] LSB

SIN EFECTO

Ejemplo:

POSICIÓN	LENG. MÁQUINA	LENG. ASSEMBLY
8F00	F0 16	BEQ $16

	ANTES		DESPUÉS
Contador programa	02	lo	18
	8F	hi	8F

Memoria programas: F0 ($8F00), 16

JR Z — SALTO RELATIVO EN CERO — Z80

Relativo 28 (2 bytes)

El valor del byte que sigue al opcode desplaza el contenido del contador del programa.

EFECTO EN EL PSR

S Z H V N C

MSB [| | | | | | |] LSB

SIN EFECTO

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBL
8F00	28 16	JR Z,$16

	ANTES		DESPUÉS
Contador programa	02	lo	18
	8F	hi	8F

Memoria programas: 28 ($8F00), 16

INX — INCREMENTAR REGISTRO X — 6502

Implícito E8 (1 byte)

El contenido del registro X se incrementa en una unidad.

EFECTO EN EL PSR

S V B D I Z C

MSB [X| | | | | |X|] LSB

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
F391	E8	INX

	ANTES	DESPUÉS
PSR	????????	0?????1?
X	FF	00

Memoria programas: E8 ($F391)

INC IX — INCREMENTAR IX — Z80

Implícito DD23 (2 bytes)

El contenido de IX se incrementa en una unidad.

EFECTO EN EL PSR

S Z H V N C

MSB [| | | | | | |] LSB

SIN EFECTO

Ejemplo:

POSICIÓN	LENG. MÁQUINA	ASSEMBLY
F391	DD23	INC IX

	ANTES		DESPUÉS
Contador programa	FF	lo	00
	E7	hi	E8

Memoria programas: DD ($F391), 23



Un gran porvenir

Desde su introducción, el CP/M se ha convertido en el estándar de la industria para sistemas operativos

Gary Kildall, uno de los miembros del equipo de Intel que desarrolló el microprocesador 8080, creó su primera versión del sistema CP/M en 1974 para apoyar a un compilador para PL/M, el primer lenguaje de alto nivel que produjo Intel. En 1975 agregó un editor (ED), un ensamblador (ASM) y un *debugger* (DDT), o programa de depuración de programas usuario. Le ofreció el nuevo sistema operativo a Intel, que lo rechazó, hecho que probablemente haya sido lo mejor que le pudiera haber ocurrido a Kildall. Asociado con Dorothy McEwan, empezó a publicar revistas para aficionados y a vender el CP/M de forma particular. El CP/M de Kildall superó rápidamente en ventas a las revistas para aficionados.

Ya sea por el diseño o debido a un rotundo golpe de suerte, Kildall había dado con un sistema que solucionaba en gran medida el principal problema del microordenador en sus primeros años: el de la compatibilidad. Los tres ordenadores de mayor consumo más importantes de finales de los años setenta (el PET, el Apple y el Tandy) tenían sistemas operativos de disco incompatibles, y los productores independientes de software tenían que optar por un formato u otro. El código debía reescribirse por completo para lograr que un producto de software funcionara con una máquina diferente de aquella para la cual había sido diseñado. Pero el CP/M vino a cambiar completamente esta situación: su considerable popularidad significó que una gran mayoría de fabricantes comenzara a adoptarlo, creando por consiguiente un "estándar" de facto. Muchas firmas que habían elegido para sus máquinas los procesadores Intel 8080 o Zilog Z80 especificaron el CP/M porque ofrecía una forma sencilla de manejar el acceso a pantalla, impresora, discos, etc. Su popularidad iba en aumento y cada vez se disponía de más software CP/M, lo que representaba un incentivo aún mayor para adoptarlo.

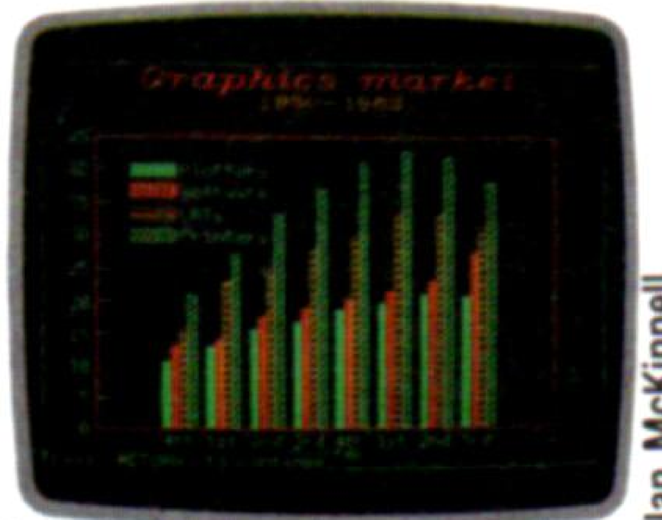
El Programa de Control para Microprocesadores al principio se autorizó para unos pocos usuarios selectos. La ahora famosa abreviatura inicialmente respondía a "Control Program/Monitor", ¡pero este título tan humilde se cambió enseguida! Hacia 1976, Kildall estaba abrumado por los pedidos que se le hacían del producto. Renunció como profesor de informática en una escuela de Monterey y fundó Digital Research en Pacific Grove (California).

Mientras el CP/M se difundía, Digital Research se concentró en los sistemas para múltiples usuarios y produjo el MP/M. Éste estaba diseñado para ser compatible con el CP/M en todos los aspectos, si bien sus primeras versiones no compartieron nada el éxito del CP/M. El fraccionamiento de las áreas de usuario y otras configuraciones que el programador de sistemas pudiera necesitar hacer no eran en absoluto sencillas y en el tratamiento de archivos difería a veces del CP/M. No obstante, debido a que los costos de los microprocesadores han disminuido a medida que ha ido aumentando la producción, la necesidad de que varios usuarios compartan un procesador ya no tiene ningún sentido desde el punto de vista económico, y el ahora revisado MP/M no ha logrado hacerse popular.

Digital Research reunió fondos de varias empresas de inversión en 1981, para convertirse en una auténtica multinacional, con una presencia notablemente fuerte en Europa (donde tiene oficinas en Gran Bretaña, Alemania y Francia). Aproximadamente al mismo tiempo, Digital Research fue una de las primeras compañías en asegurarse el contrato para desarrollar un sistema operativo para el re-



Diseños LOGO
Digital Research ha entrado en el campo de los lenguajes y está en el primer lugar con su DR LOGO. Uno de sus puntos fuertes lo constituyen sus gráficos



Ian McKinnell

Gráficos de gestión
GXS es un paquete de software pionero diseñado para lograr que las aplicaciones de gestión sean portables entre máquinas diferentes, como el paquete para gráficos de gestión que vemos en la ilustración



Cortesía de Digital Research Inc.

John Rowley, presidente de Digital Research Incorporated



Gary Kildall



Digital Research Incorporated, Massachusetts (Estados Unidos)

Cortesía de Digital Research Inc.

cientemente diseñado Personal Computer de IBM. A pesar de que el contrato para el IBM-PC al final lo obtuvo Microsoft, Digital Research no salió para nada derrotada. Desde entonces actualizó el CP/M para los procesadores de Intel 8088/8086 para hacerlos similares al MS-DOS y también ha dado un nuevo paso adelante con el Concurrent CP/M.

El Concurrent CP/M es el opuesto del MP/M, permitiendo la ejecución simultánea de diversos programas en el mismo procesador. Con este programa, un usuario podría trabajar en tres tareas diferentes al mismo tiempo (p. ej., hoja electrónica, generación de informes y correo electrónico) pasando de una a otra a voluntad. Las versiones existentes del Concurrent CP/M pueden visualizar simultáneamente cada pantalla (o parte de ella) utilizando una configuración de "ventana". Nuevas versiones del Concurrent CP/M prometen ejecutar directamente la mayoría de los programas escritos para el IBM PC-DOS.

Entre las decisiones estratégicas que han tomado Digital Research y muchas otras firmas de sistemas y lenguajes, está la de concentrar todo su trabajo de desarrollo en el lenguaje C, que es especialmente notable por su portabilidad. El código escrito en C sólo debe ser recompilado para que pueda ser usado en otro procesador, aunque esta característica ha despertado críticas en cuanto a que se trata de una codificación incómoda. Es mejor, razonan sus detractores, hacer un trabajo adecuado en ensamblador para cada procesador individual. Sin embargo, ha ido adquiriendo una creciente popularidad, y puesto que el sistema operativo UNIX, tan ampliamente utilizado, está escrito en C, la tendencia hacia este lenguaje parece irreversible.

Digital Research ha sido ciertamente coherente con su idea de que la verdadera portabilidad sólo es posible a través de lenguajes de alto nivel. Ahora proporciona varios lenguajes para una amplia gama de micros. En el extremo más bajo del mercado, sin embargo, Digital Research ha organizado un Departamento de Productos para el Consumidor que venderá BASIC Personal, CP/M Personal y su propia versión de LOGO. El Personal CP/M, al igual que el CP/M-86, está diseñado para ser almacenado en ROM y pronto estará disponible en un chip Z80 en virtud de un acuerdo con Zilog. Digital Research describe esto como *microware* y está segura de prolongar la vida activa de un gran número de programas CP/M "estándar" en vías de quedar obsoletos haciendo que sean suficientemente baratos para el usuario de ordenadores personales.

Posteriores desarrollos que prometen son VIP y GSX. VIP es una "concha" barata que permite que quienes desarrollan programas puedan presentar una interface uniforme para el usuario, independientemente del paquete de aplicaciones que se esté ejecutando. Varias aplicaciones pueden compartir los mismos datos, y éstos pueden ser transferidos de una a otra. En este sentido el VIP es similar a la tecnología Lisa de Apple y Macintosh pero requiere mucha menos memoria. El VIP puede funcionar en cualquier ordenador con más de 50 Kbytes de RAM y equipado con 150 Kbytes o más de espacio de disco.

El GSX está destinado a ser para los gráficos lo que el CP/M es para los discos. Utiliza un juego estándar de funciones para gráficos que se pueden emplear en una variedad de distintos elementos de hardware. Un programa GSX funciona en una pantalla en color, una pantalla en blanco y negro, una impresora matricial y un plotter, sin ninguna clase de modificaciones. Sin embargo, Digital Research está teniendo dificultades con la creación de un nuevo estándar para gráficos, porque el sistema no produce la misma calidad que los programas que están escritos especialmente para una máquina. Su popularidad también ha sufrido la influencia negativa de la carencia de software.

Digital Research se ha establecido como una de las principales casas de software en el negocio del microordenador. Y, sin embargo, no se está durmiendo en los laureles. Después de su incursión en productos tales como el GSX, el VIP y el LOGO, tiene potencial para seguir a empresas como Microsoft en el campo del software para aplicaciones. Con el insuperable antecedente del éxito del CP/M, la empresa parece preparada para un largo futuro.

mi COMPUTER

Con este fascículo se han puesto a la venta las tapas correspondientes al tercer volumen.



El juego de tapas va acompañado de un sobre con los transferibles, numerados del 1 al 8, correspondientes a los volúmenes de que consta la obra; esto le permitirá marcar el lomo de cada uno de los volúmenes, a medida que aumente su colección.

Para encuadernar los 12 fascículos que componen un volumen, es preciso arrancar previamente las cubiertas de los mismos.

No olvide que, antes de colocar los fascículos en las tapas intercambiables, debe usted estampar el número en el lomo de las mismas, siguiendo las instrucciones que se dan a continuación:



1 **Desprenda la hojita de protección y aplique el transferible en el lomo de la cubierta, haciendo coincidir los ángulos de referencia con los del recuadro del lomo.**

2 **Con un bolígrafo o un objeto de punta roma, repase varias veces el número, presionando como si quisiera borrarlo por completo.**

3 **Retire con cuidado y comprobará que el número ya está impreso en la cubierta. Cúbralo con la hojita de protección y repita la operación anterior con un objeto liso y redondeado, a fin de asegurar una perfecta y total adherencia.**

Cada sobre de transferibles contiene una serie completa de números, del 1 al 8, para fijar a los lomos de los volúmenes. Ya que en cada volumen sólo aplicará el número correspondiente, puede utilizar los restantes para hacer una prueba preliminar.

Editorial Delta


10036
9 788485 822836